INFORMATICA

DORIN IONIŢĂ CÂRSTOIU

SISTEME EXPERT

ALL

Sisteme expert

conf. dr. ing. **Dorin Cârstoiu**
Universitatea "Politehnica", Bucureşti

Editura **ALL**
Bucureşti

PREFAȚĂ

Sistemele expert sînt produse ale Inteligenței Artificiale, ramură a ştiinței calculatoarelor ce urmăreşte dezvoltarea de programe inteligente. Cu toate că existența marchează peste 25 ani, ele devin populare abia în ultimii ani ca urmare a scăderii costurilor informațiilor tehnologice şi a calculatoarelor, motiv pentru care numărul utilizărilor potențiali este într-o continuă creştere. Ceea ce este rematcabil pentru sisteme expert, este aria de aplicabilitate ce a cuprins multe domenii de activitate de la arhitectură, bănci, comerț, educație, ingineria sistemelor, etc.

Această carte nu favorizează un domeniu în detrimentul altuia, ea încearcă să surprindă elemente de bază în dezvoltarea sistemelor expert, metodologii de lucru, clase de aplicații. Cartea este structurată în zece capitole ce nu pot fi parcurse independent, fiecare fiind o parte din ansamblul de concură la devoltarea unui astfel de sistem.

Capitolul 1 se doreşte o introducere în domeniu şi totodată o definire a noțiunii de sistem expert. Sînt analizate principalele sisteme expert cunoscute subliniindu-se în principal particularitățile lor.

Capitolul 2 are ca obiectiv introducerea cititorului în teoria limbajelor formale şi a logicii matematice, cu accente pe limbajele de ordinul întîi, a teoriei inferențelor logice şi logicii vagi.

Capitolul 3 descrie principalele componenete ale unui sistem expert, arhitecturi complete, sistem esențial, structuri de sisteme expert destinate activităților inginereşti. O atenție deosebită este acordată sistemelor în timp real, sisteme pentru care timpul este un parametru critic.

Metodele de reprezentare a cunoaşterii formează obiectul capitolului 4. Sînt analizate aici atît activitățile privind organizarea, clasificarea şi metareprezentarea, cît şi principalele moduri de reprezentare a cunoaşterii. Întrucît performanțele oricărui sistem expert sînt dependente de conținutul şi organizarea bazei de cunoştințe, se au în vedere algoritmi de organizare a acesteia în scopul simplificării strategiei de control.

O mare atenție este acordată sistemului rezolutiv. Acesta face obiectul mai multor capitole, capitolul 5 tratînd problematica generală a sistemului

O mare atenție este acordată sistemului rezolutiv. Acesta face obiectul mai multor capitole, capitolul 5 tratînd problematica generală a sistemului rezolutiv, tipuri de strategii de control, reprezentarea problemelor, metoda rezoluției, reprezentarea în spațiul stărilor. Capitolul 6 este destinat strategiilor de căutare şi tratează în principal metode de căutare în arbori şi grafuri. Pentru fiecare metodă de căutare sînt prezentați în pseudocod algoritmii utilizați, se dau demonstrații de consistență şi convergență a acestora. În capitolul 7 se tratează metodele de rezolvare a problemelor prin decompoziție, avînd ca principal obiectiv tratarea metodelor de căutare în grafuri ŞI/SAU. Controlul rezolvării problemelor, utilizarea experienței, planuri ierarhizate fac obiectul capitolului 8. Pentru susținerea rezultatelor teoretice un mare număr de exemple este prezentat.

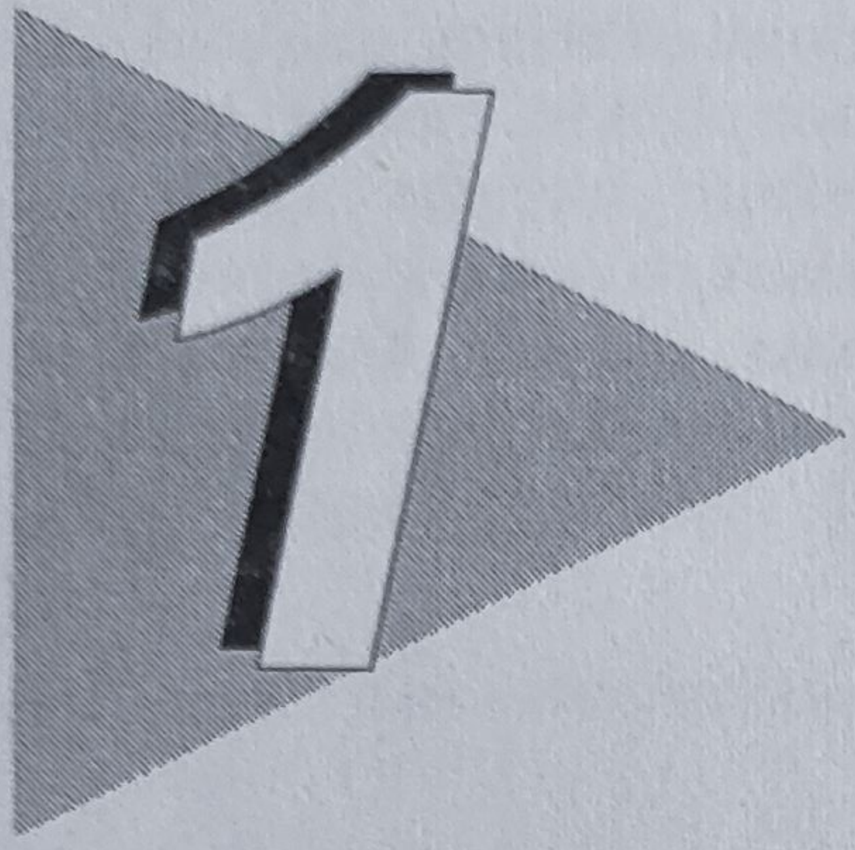
Capitolul 9 încearcă să surprindă categoriile de aplicații pentru a căror rezolvare sînt necesare sisteme expert, cît şi fazele şi metodologia de realizare a acestora.

În final, sînt prezentate trei aplicații simple, dar cu înalt grad de generalitate ce nu utilizează un sistem cadru pentru dezvoltare. Sursele programelor scrise sînt prezentate în speranța ca cititorii vor fi mai mult atraşi de acest domeniu interdisciplinar şi de mare actualitate.

Aş dori să mulțumesc pe această cale colectivului de cadre didactice al facultății de Automatică şi Calculatoare, care a avut un rol important în formarea mea profesională. Vii mulțumiri le sînt adresate studenților mei ce au contribuit nemijlocit la definitivarea materialului prin discuțiile purtate. În mod special aş dori să evidențiez absolvenții Hagiu Daniela, Anghel Alice şi Papacica Adrian care au contribuit la realizarea aplicațiilor prezentate în capitolul 10.

Lucrarea este destinată unei categorii largi de cititori, indiferent dacă sînt specialişti în calculatoare, întrucît problemele şi tehnicile Inteligenței Artificiale şi în special ale Sistemelor Expert au fascinat un mare număr de cercetători din domenii disciplinare foarte variate. Textul poate fi utilizat şi de material pentru cursul Sisteme Expert în Automatica, predat studenților din anul IV ai facultății de Automatică şi Calculatoare.

CUPRINS



INTRODUCERE

Inteligenţa Artificială este o ramură a ştiinţei promovată recent în universităţi şi laboratoare de cercetare. Astfel de tehnici au început să fie promovate cu succes şi în industrie. Se apreciază ca în viitor tratarea problemelor inginereşti cu metode specifice inteligenţei artificiale va creşte considerabil. În plus, tehnicile dezvoltate pe baza programării clasice dau rezultate corespunzătoare cînd se utilizează în descrierea elementelor finite, la simularea circuitelor, prelucrarea problemelor algoritmice, nefiind adecvate la o serie de probleme inginereşti. Metodele inginereşti pot fi caracterizate după B. V. Koen - 1985 prin "utilizarea algoritmilor euristici în scopul determinării celei mai bune soluţii într-o situaţie concret dată".

1.1. Istoria inteligenţei artificiale

Antecedentele IA trebuiesc căutate la vechii greci Gorder 1986, dar ideea formării unei discipline de cercetare a inteligenţei umane şi a raţionării formale a condus diverse grupuri din S.U.A. şi Europa după cel de-al doilea război mondial. Cercetările includ pionieratul în cibernetică şi teoria calculului. Ideea de bază în cibernetică trebuie să se bazeze pe principiul "reacţiei negative" Wiener 1948, prin care se controlează modul în care fenomenele fizice pot fi generalizate pentru formarea bazei gîndirii umane. Alte argumente pentru necesitatea accesului la comportarea umană este bazată pe luarea deciziilor raţionale. În conjuncţie cu aceste investigaţii, devine imposibil de exprimat o formulare a tehnicilor analitice cantitative despre aceste operaţii de cercetare. Evoluţia ştiinţei calculatoarelor şi comportarea umană a stimulat interesele raţionării pe maşină, deoarece nu oferea mijloace ce puteau fi utilizate în modelele de calcul ale gîndirii şi raţionării. Aceasta a determinat o nouă disciplină, cea a inteligenţei artificiale. Fără procedurile de calcul, nu se pot ilustra conceptele raţionării pe maşină utilizînd hîrtie, creion şi modele fizice. Ca potenţialul raţionării pe

o maşină să poată fi realizat, este necesară reunirea eforturilor de cercetare din mai multe discipline formale. Prin cercetarea începută în 1956 şi cu evenimentul consolidării prin Summer Research Project realizat la Darthmounth College, John McCarty a stabilit termenul de IA. Primele succese sînt datorate programelor capabile să demonstreze teoreme ale logicii matematice (1963) ce au determinat noua deschidere către programe ce folosesc tehnici de IA şi o revigorare a domeniului IA, devenind astfel o disciplină, mai mult decît o căutare matematică pentru maşinile inteligente. În timp, a apărut şi ideea benefică a înglobării filozofilor, lingviştilor şi neurologilor în aceste domenii.

Se pune întrebarea firească asupra căror zone ştiinţifice se extinde IA. Se dau mai jos cîteva exemple:

- vedere artificială - ce presupune recunoaşterea formelor, identic cu vederea umană;
- robotica - focalizează producerea dispozitivelor mecanice capabile să reproducă mişcarea;
- prelucrarea vocii - ce priveşte constituirea şi sinteza vocii umane;
- prelucrarea în limbaj natural - înţelegerea şi vorirea în limbaj natural;
- demonstrarea (producerea) teoremelor - în matematică şi logică;
- "General Problem Solving" - rezolvarea unei clase generale de probleme exprimate în limbaje formale;
- recunoaşterea formelor - recunoaşterea şi clasificarea diferitelor forme;
- teoria jocurilor;
- învăţarea automată - maşini ce acumulează cunoştinţe prin observarea exemplelor.

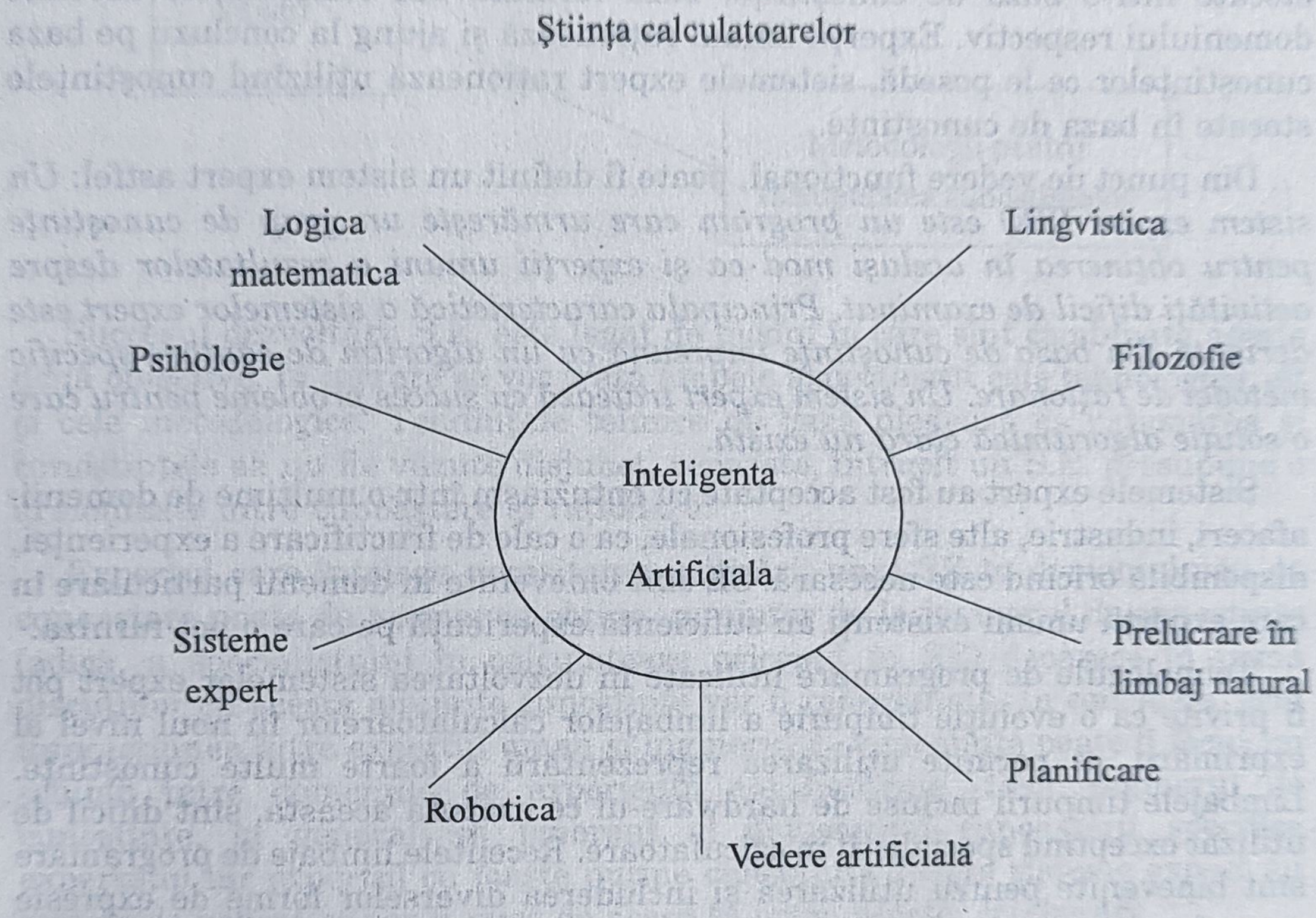
Investigaţiile în domenii cum sînt recunoaşterea formelor şi prelucrarea vocii sînt noi domenii în IA. Nu trebuie să surprindă faptul că utilizarea în ultimii ani a sistemelor expert a fost asociată cu cercetările de IA deoarece structura şi proprietăţile sînt apropiate de produsele IA.

SE beneficiază de cercetările ce au fost realizate în diferite zone ale IA. În fig. 1.1. se arată relaţiile între sistemele expert şi celelalte domenii ale IA. În prezent SE sînt tot mai mult asociate direct cu modurile de raţionare şi cunoştinţele reprezentate. SE sînt de fapt dezvoltări în noile cîmpuri ale achiziţiei de cunoştinţe ca forme specializate de învăţare, în care cunoştinţele sînt achiziţionate direct de la expert. Alte SE ca tehnici de IA includ explicaţii, învăţare inteligenţa, planificare, rezolvarea problemelor distribuite, cercetări ce sînt adresate direct IA.

Cercetările au fost în general realizate în diferite lucrări pentru doctorate ce au avut mai degrabă scopul demonstrării tehnicilor particulare şi a conceptelor, fiind mai puţin preocupate de obţinerea unor aplicaţii particulare.

O mare gamă a cercetărilor urmăresc obţinerea minimului în strategii minimax. În problemele privind parcurgerea combinaţională a arborilor de căutare, un număr mare de algoritmi au fost dezvoltaţi pentru o căutare cît mai eficientă.

▼ ***Fig. 1.1 Domenii ale inteligenţei artificiale***



Algoritmii se pot grupa după două aspecte:

- construcţia unui arbore de căutare;
- izolarea şi căutarea doar într-o parte a arborelui de căutare.

Cei mai mulţi algoritmi de acest tip au caracter euristic. O regulă euristică este o regulă ce oferă o metodă de rezolvare a problemelor, sau o metodă de căutare. Ea nu dă rezultate corecte la orice moment de timp şi nu este garantată că găseşte cea mai bună soluţie, dar în acelaşi timp poate reduce timpul de căutare. Metodele de acest tip pot fi folosite pentru reducerea mărimii arborilor de căutare în cazurile arborilor foarte mari. Metodele euristice sînt baza construcţiei pentru GPS (General Problem Solver).

1.2. Sisteme expert, definiţii

O ramură a Inteligenţei Artificiale (IA) este reprezentată şi de către sistemele expert. Un sistem expert este un program care urmăreşte cunoştinţele, raţionează pentru obţinerea rezultatelor într-o activitate dificilă întreprinsă uzual doar de experţii umani. Dacă un expert uman are cunoştinţe într-un domeniu specific, un sistem expert utilizează cunoştinţele ce sînt stocate într-o bază de cunoştinţe, bază formată din cunoştinţele asociate domeniului respectiv. Experţii umani raţionează şi ajung la concluzii pe baza cunoştinţelor ce le posedă, sistemele expert raţionează utilizînd cunoştinţele stocate în baza de cunoştinţe.

Din punct de vedere funcţional, poate fi definit un sistem expert astfel: *Un sistem expert (SE) este un program care urmăreşte un grup de cunoştinţe pentru obţinerea în acelaşi mod ca şi experţii umani a rezultatelor despre activităţi dificil de examinat. Principala caracteristică a sistemelor expert este derivată din baza de cunoştinţe împreună cu un algoritm de căutare specific metodei de raţionare. Un sistem expert tratează cu succes probleme pentru care o soluţie algoritmică clară nu există.*

Sistemele expert au fost acceptate cu entuziasm într-o mulţime de domenii: afaceri, industrie, alte sfere profesionale, ca o cale de fructificare a experienţei, disponibilă oricînd este necesară. SE sînt binevenite în domenii particulare în care experţii umani existenţi au suficientă experienţă pe care o pot furniza.

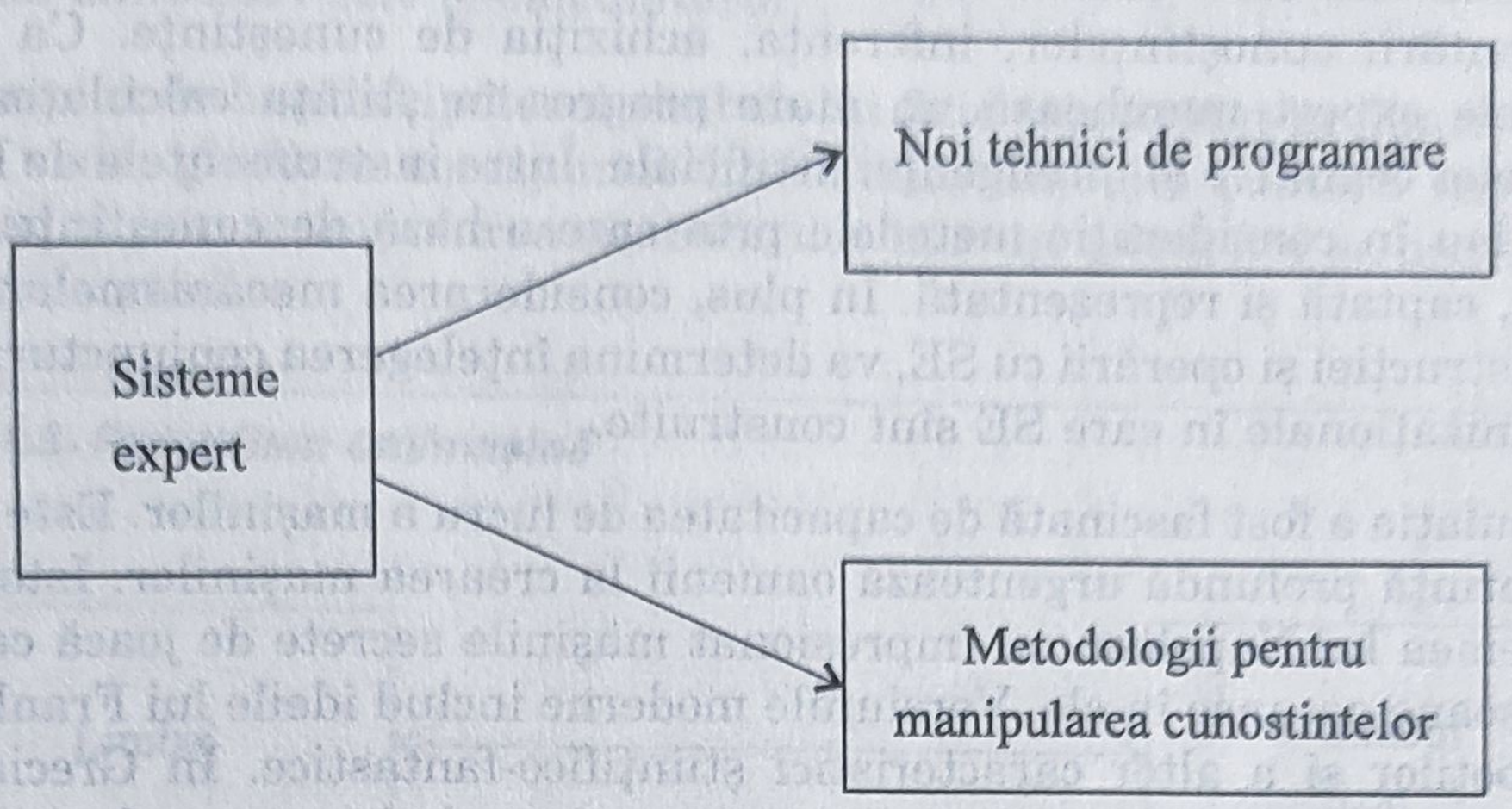
Tehnologiile de programare utilizate în dezvoltarea sistemelor expert pot fi privite ca o evoluţie timpurie a limbajelor calculatoarelor în noul nivel al exprimării, ce permite utilizarea reprezentării a foarte multe cunoştinţe. Limbajele timpurii incluse de hardware-ul ce executa aceasta, sînt dificil de utilizat exceptînd specialiştii în calculatoare. Recentele limbaje de programare sînt binevenite pentru utilizarea şi includerea diverselor forme de expresie umană.

Tehnologiile de programare specifice SE marchează alţi paşi în această direcţie. Pentru a fi utilizată efectiv, fiecare tehnologie nouă este însoţită de un set de ghiduri şi metodologii pentru utilizare. Aceasta ajută noii utilizatori la creşterea eficienţei utilizării şi a maximizării productivităţii. Din acest motiv utilizarea tehnicilor specifice în SE cu foarte multe cunoştinţe va necesita un nou set de metodologii, bazate pe experienţa crescută, în dezvoltarea aplicaţiilor.

Aşa cum se arăta în fig. 1.2 un SE este bazat pe două componente distincte complementare:

a) noi tehnologii de programare ce permit utilizarea a foarte multe cunoştinţe, precum şi modul de inferenţiere cu acestea;
b) noi construcţii şi metodologii dezvoltate, ce permit utilizarea efectivă a acestor tehnologii legate cu probleme complexe.

▼ ***Fig. 1.2. Componentele de bază ale SE***



Succesul dezvoltării S.E. este legat de modul în care sînt combinate aceste două obiective. În lucrare se vor trata ambele aspecte atît cele tehnologice, cît şi cele metodologice. Tendinţele tehnice de bază pledează ca raţionarea şi cunoştinţele să nu fie văzute disjunct, separate, întrucît un S.E. presupune o armonizate între cunoaştere şi raţionare.

Expertul care înţelege necesitatea utilizării unui SE în domeniul său de cunoaştere poate de asemenea obţine un ajutor de la inginerul de cunoştinţe (adică, a specialistului în calculatoare priceput în SE) deoarece în urma discuţiilor cu acesta anumite cunoştinţe vor fi reformulate. În concluzie, prin interacţiunea între expertul uman şi inginerul de cunoştinţe poate fi făcut un schimb între domeniile de experienţă ale fiecăruia. Deci, inginerul de cunoştinţe, în general un ignorant al domeniului cunoaşterii specifice expertului iar expertul cu foarte puţine cunoştinţe despre sisteme expert au puncte de vedere ce vor fi puse de acord în urma acestei colaborări.

În concluzie, nu este uşor pentru un expert să găsească un inginer de cunoştinţe în vederea colaborării deoarece cunoştinţele sînt expansive şi greu de formulat într-o manieră concisă. Aceasta motivează necesitatea ca experţii umani să aibă mai multe cunoştinţe despre sisteme expert. Sînt necesare cît mai multe utilitare specifice dezvoltării sistemelor expert, trebuie să fie posibil ca eventual mai mulţi experţi să fie adepţii utilizării SE fără a fi utilizate cunoştinţele lor în acelaşi timp de către inginerii de cunoştinţe. În orice eventualitate punctele de vedere ale inginerului de cunoştinţe sînt utilizate în proiectare şi este limpede ca procesul construirii sistemelor expert este mai eficient dacă expertul este cunoscător asupra tehnologiilor şi metodologiilor sistemelor expert.

Sistemele expert fac apel la o serie de mecanisme de înalt nivel cu dedicaţie specială fără a se face apel la limbajele clasice cum sînt C şi Pascal, mecanismele specifice permiţînd implementarea noilor tipuri de exprimări.

Conştientizarea experţilor despre sistemele expert şi cunoaşterea inginerească este de un mare beneficiu pentru dezvoltarea SE. Pentru facilitarea acestora este esenţial să se distingă elementele esenţiale ale reprezentării cunoştinţelor, inferenţa, achiziţia de cunoştinţe. Ca urmare, sistemele expert marchează un mare progres în ştiinţa calculatoarelor, a sistemelor cognitive şi inteligenţei artificiale. Între instrumentele de lucru ale SE se iau în consideraţie metodele prin care o bază de cunoştinţe poate fi afişată, captată şi reprezentată. În plus, considerarea mecanismelor de bază ale construcţiei şi operării cu SE, va determina înţelegerea conjuncturii sociale şi organizaţionale în care SE sînt construite.

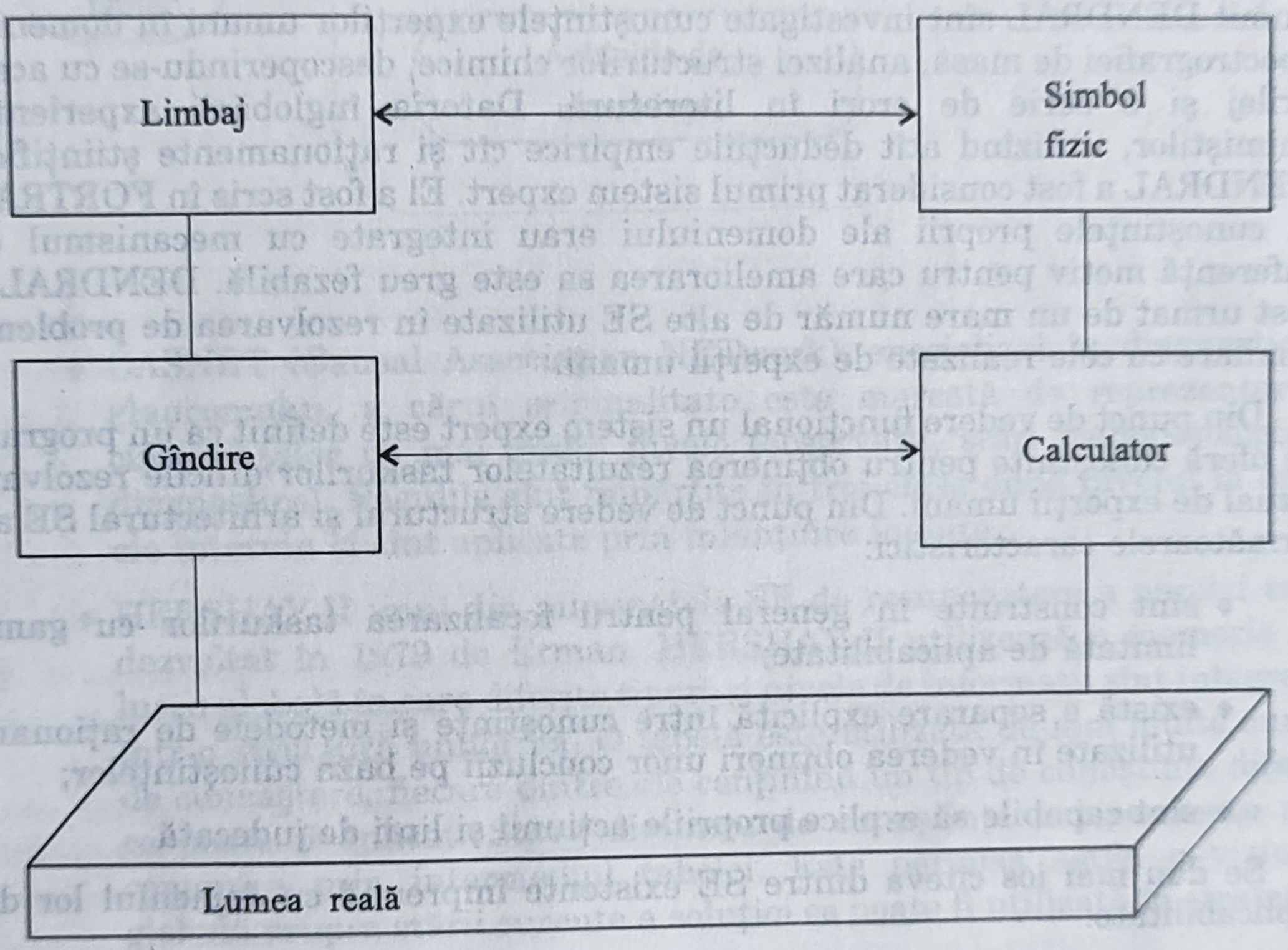
Populaţia a fost fascinată de capacitatea de lucru a maşinilor. Este evident, ca o dorinţă profundă urgentează oamenii la crearea maşinilor. Istoric, încă din vremea lui Napoleon au impresionat maşinile secrete de joacă ce operau cu persoane ascunse în ele. Versiunile moderne includ ideile lui Frankenstein ale roboţilor şi a altor caracteristici ştiinţifico-fantastice. În Grecia antică filozofii au propus crearea unor mecanisme formale bazate pe logică. Cîteva variante pornesc de la construcţia modelelor ce furnizează raţionarea mecanizată. Un exemplu este Ramon Lull din Spania, care a construit o maşină ce urmăreşte demonstrarea faptului că Dumnezeu există. Lull utilizează literele pentru reprezentarea cuvintelor şi argumentelor, precum şi combinaţii ale acestor simboluri pe baza unui sistem de reguli. La baza sistemului construit de Lull stă o schemă mecanică de construcţie a unei varietăţi de figuri geometrice, care rotite unele faţă de altele determină noi argumente logice. Sistemul său este limitat de stadiul de dezvoltare al geometriei tridimensionale prin numărul operaţiilor geometrice posibile. Atingerea scopului propus de SE a fost imposibil de realizat pînă la dezvoltarea calculatoarelor moderne şi desăvîrşirii noii revoluţii intelectuale. La capacităţile tehnologice actuale mimarea capabilităţilor umane este o necesitatea ca maşinile inteligente să devină o realitate.

Principala virtute a calculatoarelor este abilitarea de realizare a unei mari viteze de calcul. Fiecare limbaj de programare încearcă să facă calculatorul cît mai eficient din punctul de vedere al puterii de calcul. Calculatoarele încep să se extindă la aplicaţiile inginereşti şi bancare, persoanele interesate de calculatoare încep să utilizeze termenul de sistem de procesare. În plus calculul matematic, sistemele automate, sînt realizate prin intermediul unor programe de calcul ce transpun în maşină gîndirea omului ce le-a programat. Pe măsură ce cercetările evoluează, se dezvoltă aşa-zisele programe "elegante". Principalul în aceste activităţi, este creşterea interesului intelectual, cel comercial rămînînd pe locul secund. Oricînd ne gîndim la programele de interes, trebuie ştiut faptul că programele de acest tip determină creşterea gîndirii, în consecinţă, demonstraţii interesante au fost construite fără ca aplicaţiile să fi fost dezvoltate. Problemele ce se solicită rezolvate prin sisteme expert fac apel la gîndire, la abilitatea de raţionare cu alte concepte decît numere, problematica ce devine din ce în ce mai mult o necesitate. Realizarea programelor ce raţionează porneşte de la faptul că

simbolurile de prelucrare pot fi numere, texte, sau alte concepte. Aceasta duce la "ipoteza simbolului fizic" ca simbol fizic esenţial. O maşină ce poate manipula simboluri este şi calculatorul.

Principiul de bază al ipotezei simbolului fizic este arătat în figura 1.3. Aşa cum se vede, gîndirea corespunde limbajului, limbajul poate fi captat prin simboluri şi prin manipularea simbolurilor într-un calculator se poate simula procesul de gîndire.

▼ ***Fig. 1.3. Paralelism om/maşină***



Pornind de la ipoteza simbolului fizic şi cu "bunăvoinţa construcţiei maşinilor cu viaţa", cele mai multe cercetări în inteligenţa artificială încearcă să rezolve problema reprezentării în limbaje de calcul simbolic. În sprijinul acestui entuziastm şi inerentelor dificultăţi ale raţionării pe maşină au fost realizate o serie de cercetări. O problemă serioasă asupra limitei capacităţii de memorare şi a puterii de calcul se punea în faza iniţială a epocii informatice. În urma cercetărilor, utilizînd calculatoare din ce în ce mai performante, problema raţionării a putut fi rezolvată.

1.3. Originea Sistemelor Expert

După ce au fost trecute limitările de spaţiu şi timp, apreciate la cîteva ordine de mărime, au fost marcate succese în comercializarea mai multor programe ce furnizează taskuri specifice cu cele ale experţilor umani, numite şi sisteme expert. În 1960 NASA a hotărît trimiterea unui vehicul pe Marte, unul dintre scopuri fiind cercetarea structurii chimice a solului acestei planete. Pentru aceasta cercetătorii de la Stanford au construit un program ce a trebuit să înglobeze experienţa chimiştilor în identificarea structurii chimice a unor substanţe pornind de la spectrograma de masă. În dezvoltarea programului numit DENDRAL sînt investigate cunoştinţele experţilor umani în domeniul spectrografiei de masă, analizei structurilor chimice, descoperindu-se cu acest prilej şi o serie de erori în literatură. Datoria înglobării experienţei chimiştilor, utilizînd atît deducţiile empirice cît şi raţionamente ştiinţifice, DENDRAL a fost considerat primul sistem expert. El a fost scris în FORTRAN şi cunoştinţele proprii ale domeniului erau integrate cu mecanismul de inferenţă motiv pentru care ameliorarea sa este greu fezabilă. DENDRAL a fost urmat de un mare număr de alte SE utilizate în rezolvarea de probleme similare cu cele realizate de experţii umani.

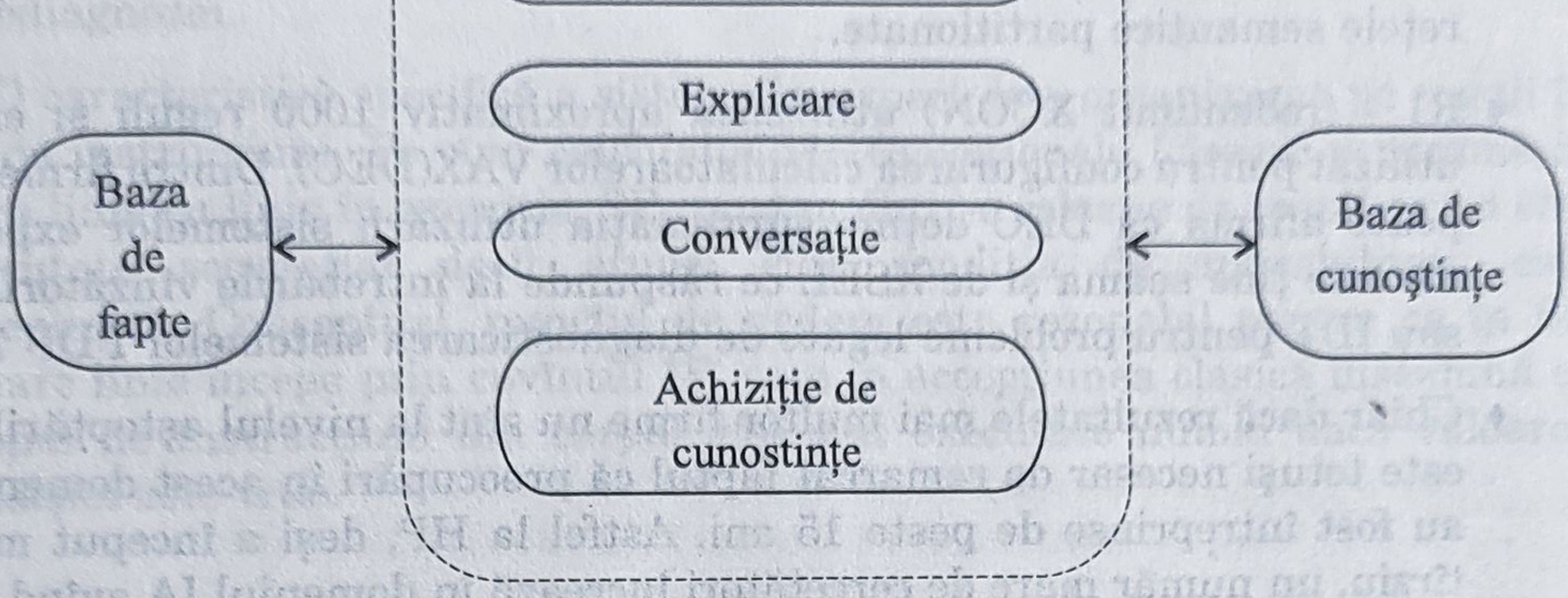
Din punct de vedere funcţional un sistem expert este definit ca un program ce oferă cunoştinţe pentru obţinerea rezultatelor taskurilor dificile rezolvate uzual de experţii umani. Din punct de vedere structural şi arhitectural SE au următoarele caracteristici:

- sînt construite în general pentru focalizarea taskurilor cu gamă limitată de aplicabilitate;
- există o separare explicită între cunoştinţe şi metodele de raţionare utilizate în vederea obţineri unor concluzii pe baza cunoştinţelor;
- sînt capabile să explice propriile acţiunii şi linii de judecată.

Se dau mai jos cîteva dintre SE existente împreună cu domeniul lor de aplicabilitate:

- MYCIN foloseşte experinţa din DENDRAL şi operează ca un consultant pentru medici în investigarea pacienţilor pentru depistarea infecţiilor bacterigene ale sîngelui şi pentru prescrierea tratamentului adecvat. Sistemul operează într-o manieră interactivă, mediul oferind toate informaţiile utile despre pacient, cît şi rezultatele de laborator. Raţionarea în MYCIN în absenţa informaţiilor complete (rezultatele de cultură nu sînt cunoscute decît după 24 pînă la 48 ore) se bazează pe un ansamblu de cunoştinţe. El este capabil să-şi explice raţionamentul, baza de cunoştinţe poate fi schimbată la orice moment de timp. MYCIN este compus dintr-o bază de cunoştinţe ce cuprindea 300 de reguli în '77, 450 în '80 şi este compus din patru secţiuni de program principale ce formează un tot unitar (fig. 1.4). El a fost scris în interlisp pe DEC PDP10 şi ocupă 130 K. Dintre aceştia 50 K sînt ocupaţi de sistemul de consultare, 20 K de sistemul de achiziţie a cunoştinţelor.

▼ *Fig. 1.4. Structura sistemului MYCIN*



- CASNET (Causal Association NETwork), specializat în diagnosticul glaucomului, a cărui originalitate este marcată de reprezentarea cunoştinţelor pe mai multe nivele (observaţii, stări psihopatologice, diagnostice). Regulile sînt împărţite în trei clase după nivelul la care ele intervin şi sînt aplicate prin înlănţuire înainte.

- HERSHAY-II unul din minunatele SE de recunoaştere a parolei este dezvoltat în 1979 de Erman. HERSHAY-II utilizează o memorie de lucru globală în care diferite tipuri şi nivele de informaţii sînt integrate într-o structură uniformă. O tabelă este utilizată de mai multe surse de cunoaştere, fiecare dintre ele conţinînd un tip de cunoştinţe diferit ce poate fi aplicat. Surse distincte de cunoştinţe independente pot comunica prin intermediul tabelei. Este permisă astfel o viziune globală asupra stării curente a soluţiei ce poate fi utilizată în strategia de control.

- INTERNIST devenit apoi CADUCEUS, dezvoltat la universitatea din Pittsburg, acoperea în anul apariţiei (1982) 80% din medicina internă. El cuprinde mai multe milioane de reguli de producţie grupate în subiecte particulare. Fiecare boală este legată de observaţii prin asociativitate în două sensuri, observaţie - diagnostic şi invers, ponderate printr-un coeficient exprimînd frecvenţa fiecărei observaţii pentru un diagnostic. Proiectul a fost dezvoltat în mai mult de zece ani într-un colectiv mare de specialişti.

- MOLGEN, dezvoltat la universitatea Stanford, specializat în biologie moleculară. Dificultatea principală a sistemului a fost legată de necesitatea integrării în baza de cunoştinţe a informaţiilor diverse de biologie, topologie, chimie, tehnologia manipulării şi genetică.

- PROSPECTOR, dezvoltat la universitatea Stanford, este specializat în prospecţiunile miniere. Prin program au fost obţinute rezultate spectaculoase în urma analizei cantităţii de molibden şi fosfor exploatate din muntele Tolmar, în legătură cu descoperirea altor zăcăminte. Acesta este unul dintre rarele sisteme ce utilizează reţele semantice, baza de cunoştinţe conţine alături de reguli de producţie, reţele semantice partitionate.

- R1 - (redenumit XCON) utilizează aproximativ 1000 reguli şi este utilizat pentru configurarea calculatoarelor VAX(DEC). Dintre firme se poate afirma că DEC deţine supremaţia utilizării sistemelor expert dacă se ţine seama şi de XSEL ce răspunde la întrebările vînzătorilor sau IDT pentru probleme legate de diagnosticarea sistemelor PDP 11.

- Chiar dacă rezultatele mai multor firme nu sînt la nivelul aşteptărilor este totuşi necesar de remarcat faptul că preocupări în acest domeniu au fost întreprinse de peste 15 ani. Astfel la HP, deşi a început mai tîrziu, un număr mare de cercetători lucrează în domeniul IA avînd în studiu tehnologii de programare, dezvoltări de pachete software pentru aplicaţii de SE (în principal diagnoza). Honeywell s-a angajat recent în rezolvarea unor chestiuni legate de achiziţia cunoştinţelor, utilizarea cunoştinţelor, arhitecturi şi ustensile pentru dezvoltarea SE. IBM şi-a orientat cercetările pe diferite sisteme generice capabile să fie introduse în diverse aplicaţii. IBM dezvoltă mai multe generaţii de SE destinate diagnosticului produselor informatice. Se remarcă aici şi alte firme de renume ca: General Electric, Schlumberger, Litho et Mirlitho, etc.

Dezvoltarea SE cere noi specialişti pentru captarea cunoştinţelor şi exprimarea acestora sub formă de reguli. Devine necesară o nouă specializare numită "inginer de cunoştinţe", un intermediar între specialistul în calculatoare şi expertul uman. Metodologia de captare a cunoştinţelor este cunoscută sub numele de ingineria cunoaşterii. Ca urmare se poate concluziona:

- un SE poate fi expert într-o singură chestiune. Dacă se construieşte un SE ce rezolvă două probleme sînt posibile critici pentru fiecare dintre acestea;

- un SE poate să rezolve numai acele probleme pe care expertul uman le-a transmis. Dacă se construieşe un SE fără a fi consultat expertul uman acesta are cunoştinţe limitate şi deci este mai slab decît expertul uman;

- un SE nu va putea niciodată să înlocuiască omul.

O caracteristică specifică SE este marcată de faptul că acestea pot fi adaptive - caracteristică marcată prin faptul ca schimbările efectuate la diferite intervale de timp nu determină modificări ale programelor. Pentru aceasta cunoştinţele sînt ţinute sub formă de reguli care pot fi uşor citite şi

modificate de utilizator cu mici intervenţii din partea acestuia. În multe SE uzuale există o distincţie netă între cunoştinţe şi mecanismele ce manipulează aceste cunoştinţe. Prin modificarea naturii cunoştinţelor utilizate, un SE devine capabil a fi utilizat în alte zone ale aceluiaşi domeniu. De exemplu, un SE destinat diagnosticului medical, prin schimbarea cunoştinţelor medicale şi substituirea acestora de cunoştinţe inginereşti poate deveni expert în domeniul autodiagnozei.

O caracteristică specifică a sistemelor expert este organizarea pe reguli în loc de instrucţiuni. Pe cînd calculatoarele conveţionale folosesc programe ce intră linie cu linie în execuţie, SE constau dintr-o colecţie de reguli ce nu sînt executate secvenţial decît atunci cînd condiţia de aplicabilitate este îndeplinită. Conceptual, punctul de vedere este rezonabil pentru ca în SE fiecare linie începe prin cuvîntul IF, care în accepţiunea clasică înseamnă că grupul de instrucţiuni din corpul său sînt executate numai dacă valoarea condiţiei este true.

1.4. Caracteristici ale Sistemelor Expert cu aplicaţii în timp real

Dacă sistemul expert rezultat este utilizat în domeniul conducerii proceselor, Astrom şi Anton (1984) denumesc noua zonă de aplicaţii "Expert Control". Astfel, în acest domeniu se distinge un prim nivel de utilizare a unui SE la nivelul unei bucle de reglare într-un controller. El nu este numai o ustensilă pentru utilizator ci este utilizat şi în acţiuni directe asupra instalaţiei controlate. În această situaţie (bucla unitară), universul cunoaşterii este limitat. Bineînţeles că la sisteme multivariabile sau pentru analiza globală a performanţelor la nivelul unei instalaţii un sistem într-o astfel de abordare este recomandat. Trebuie făcută aici o distincţie netă între zona teoretică a cercetărilor, focalizate în obţinerea algoritmilor de conducere evoluaţi, şi practica inginerească în care algoritmi clasici sînt utilizaţi şi mai ales metodele euristice bazate pe experienţa de exploatare a tehnologiilor.

O idee de utilizare a unui SE la nivelul unei bucle de reglare porneşte de la sarcina acestui task de armonizare a algoritmilor de aplicaţie numerici la procesul fizic. Algoritmii numerici pot fi împărţiţi în general în trei grupe: algoritmi de reglare, algoritmi de identificare, algoritmi de monitorizare a celor din primele două categorii. Algoritmii de reglare pot fi de complexitate variabilă de la simplii PI, PID la algoritmii optimali. Algoritmii de identificare pot fi simpli algoritmi de estimare sau algoritmi complecşi bazaţi pe metoda celor mai mici pătrate. Supervizarea se referă la detectarea erorilor staţionare, depăşirea limitelor de alarmare, instabilitate. SE va decide în ce ordine intră în execuţie aceşti algoritmi, ce parametri şi la ce valori să fie setaţi. Rezultatul aplicării unui algoritm sporeşte cunoaşterea în jurul procesului fizic condus şi

afectează modul de realizare în viitor a problemei conducerii. În aceste condiţii unul din efectele aplicării acestei strategii este creşterea gradului de cunoaştere asupra procesului.

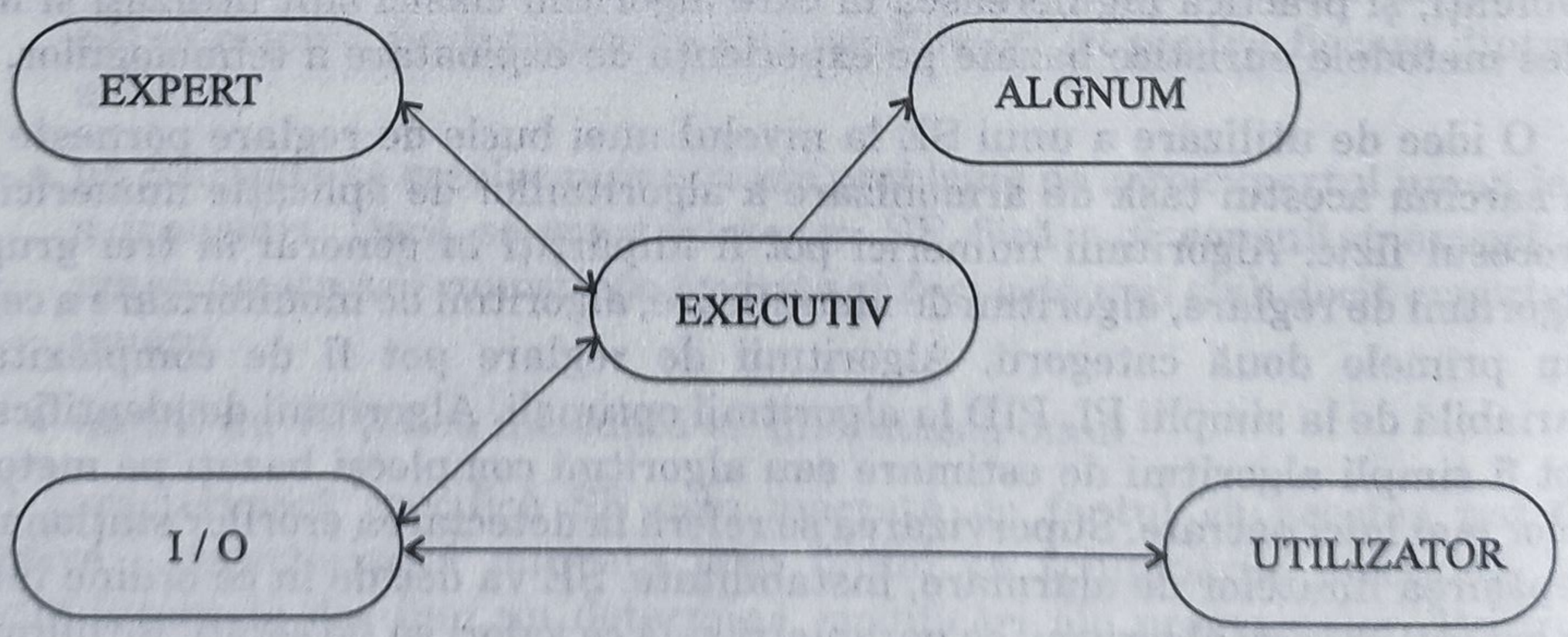
Combinarea SE, algoritmi numerici pentru reglare impune mai multe particularităţi dintre care se pot enumera:

- divizarea execuţiei: algoritmi, logica secvenţială de execuţie;
- modul de scriere a rutinelor ce implementează algoritmii numerici va fi adecvată pentru a putea fi executaţi în paralel cu sistemul expert;
- rentabilizarea codului fapt ce va simplifica dezvoltările programelor în vedere modernizării;
- rularea personalului operator de proces, cît şi creşterea transportabilităţii programelor;
- combinarea mai multor algoritmi de control, posibilitatea de modificare a perioadei de eşantionare şi de utilizare a algoritmilor adaptivi;
- schimbarea într-o manieră facilă a interfeţei utilizator;
- posibilitatea explorării adecvate a informaţiilor primare disponibile în jurul părţii fixate.

Algoritmii numerici şi sistemul expert se vor implementa ca procese paralele separate, ţinînd cont de la faptul că timpii de rulare sînt diferiţi. Dacă timpul de execuţie al algoritmului numeric este dependent de partea fixată timpul de răspuns al sistemului expert poate să fie cu cîteva ordine de mărime mai mare. Se consideră pentru exemplificare o structură cu trei procese paralele denumite (fig. 1.5):

EXPERT -partea specifică de sistem expert;
ALGNUM -biblioteca de algoritmi numerici de conducere;
COM I/0 -pachet programe pentru comunicaţie cu utilizatorul.

▼ ***Fig. 1.5. Structura unui SE la nivelul unei bucle de reglare***

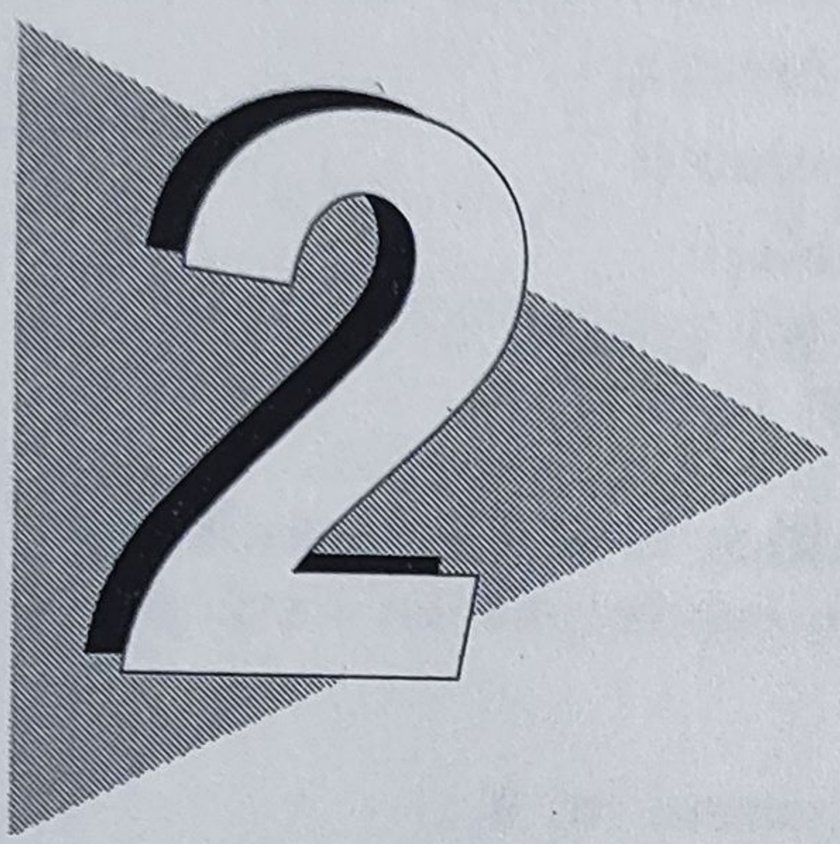


Singurul proces critic din cele descrise este ALGNUM, motiv pentru care este organizat ca task cu prioritate maximă. Cele trei procese comunică prin mecanisme de tip cutie poştală, în care mesajele sînt puse într-o coadă din care se poate citi şi în care se poate scrie. La încercarea de citire dintr-o cutie goală se va genera un apel de oprire a procesului pînă cînd mesajul soseşte. ALGNUM este un proces constituit dintr-o bibliotecă de algoritmi numerici de reglare, estimare şi supervizare. Algoritmii sînt asociaţi cu proceduri de iniţializare, execuţie, oprire şi modificare a parametrilor. Se dau mai jos exemple de mesaje ce pot fi adresate algoritmilor:

- START (algoritm, parametrii iniţiali);
- PAR (algoritm, nume parametrii, valori);
- STOP (algoritm);
- EXECUTA (algoritm); directiva ce este aferentă numai algoritmilor ce nu sînt rulaţi periodic.

Este posibilă şi schimbarea unor variabile globale pentru algoritmii de tip numeric, cum sînt: perioada de eşantionare, referinţa, mărimea de comandă. O directivă de acest tip este GLOBAL (nume parametrii, valoare).

Alte mesaje sînt transmise de la ALGNUM la EXPERT, şi au o prioritate dependentă de semnificaţie. Mesajele ce provin de la algoritmul de supervizare şi cele declanşate de nivelele de alarmare au prioritate mai mare faţă de cele transmise de algoritmii de estimare. În mod normal informaţiile circulă între instalaţia tehnologică şi ALGNUM, sistemul expert nefiind implicat atîta timp cît semnificaţia a fost bine detectată de algoritmul în lucru. Informaţiile primare extrase de către sistemul expert au ca sursă mesajele primite de la ALGNUM. O altă categorie de informaţii au ca sursă comenzile introduse de către utilizator, comenzi ce semnifică comutarea modului de operare, schimbarea parametrilor, adăugare, ştergere sau editare programe on-line.



METODE MATEMATICE ÎN INTELIGENŢA ARTIFICIALĂ ŞI SISTEME EXPERT

Este raţional a începe cu presupunerea că fenomenele din natură şi societate se desfăşoară după legi parţial cunoscute. Se porneşte de la faptul că regularitatea fenomenelor observate şi caracteristicile structurale ale acestora pot determina ordinea conceptuală, ordine ce permite abstractizarea regulilor şi legităţilor.

2.1. Limbaje formale, limbaje de ordinul întîi

Studiul matematic al sistemelor automate a dus la necesitatea descrierii formale a procesului de prelucrare a informaţiei de către calculatoare. Descrierea se face cu ajutorul unor algoritmi, înţelegînd prin aceştia o succesiune de operaţii ce conduc la un anumit rezultat. Pentru prelucrarea informaţiilor cu ajutorul calculatorelor este necesar ca omul să se facă înţeles de către acestea, motiv pentru care el trebuie să studieze toate problemele legate de teoria limbajelor formale, întrucît limbajele de programare nu sînt altceva decît limbaje formale. Intuitiv prin limbaj formal se înţelege o mulţime de şiruri de elemente dintr-un alfabet de simboluri primitive. Aceste şiruri sînt numite în mod obişnuit cuvinte sau propoziţii corecte din punct de vedere gramatical. Pentru a selecta cuvintele unui limbaj se mai consideră o mulţime finită de reguli de construcţie numită şi gramatică. În acest mod problema definirii unui anumit limbaj formal se transpune în problema definirii regulilor gramaticii care generează acest limbaj şi a indicării modului în care regulile gramaticii selectează cuvintele limbajului respectiv.

O abstractizare fundamentală este şi tipul de similaritate. Se poate defini "tipul de similaritate" ca un 5-uplu cu structura

$$\sigma = < I, J, K, \theta, \psi >$$

în care:

I - reprezintă mulţimea de indexare pentru familiile de relaţii;
J - mulţimea de indexare pentru familiile de funcţii;
K - mulţimea de indexare pentru elemente distincte;
$\theta: I \to N$ - aplicaţia de aritate a familiilor de relaţii;
$\psi: J \to N$ - aplicaţia de aritate a familiilor de funcţii, în care N este mulţimea numerelor naturale.

Utilizînd conceptul de similaritate se pot clasifica structurile, astfel ca o structură $\mathfrak{L}_\alpha$ avînd tipul de similaritate σ este un cvadruplu cu $i \in I$, $j \in J$, $k \in K$ format din:

- o mulţime A denumită şi suportul sau universul lui $\mathfrak{L}_\alpha$;
- o familie $\{R_i^\alpha \mid i \in I\}$ astfel că pentru fiecare $i \in I$, R_i^α este o relaţie pe A cu aritatea $\theta(i)$;
- o familie $\{F_j^\alpha \mid j \in J\}$ astfel că pentru fiecare $j \in J$, F_j^α este o funcţie în A cu aritatea $\psi(j)$;
- o submulţime $(e_k^\alpha \mid k \in K)$ a lui A, denumită şi mulţimea elementelor distincte din A.

În ansamblu dacă $I = \varnothing$, adică structura α nu conţine simboluri relaţionale, atunci ea se numeşte şi algebră. Dacă $J = \varnothing$, adică structura α nu conţine simboluri funcţionale, ea se numeşte sistem relaţional.

Fie ca exemplu structura

$$\alpha = < N, <=, +, *, s, 0 >$$

ce este o aritmetică a numerelor naturale în care:

<= - simbolul relaţional;
+, * - simbolurile funcţionale binare;
s - simbolul funcţional numit succesor, $s(x) = x + 1$;
0 - simbolul elementului distinct 0, o constantă (constanta zero).

Un "limbaj de ordinul întîi" $\mathfrak{L}_\sigma$ se poate asocia la orice tip de similaritate, astfel că orice enunţ din $\mathfrak{L}_\sigma$, are o valoare de adevăr definită în orice structură avînd tipul de similartitate σ. Un limbaj de ordinul întîi $\mathfrak{L}_\sigma$ este construit din: simboluri primitive, termeni, formule atomice, formule, enunţuri. Din categoria simbolurilor primitive fac parte:

a) variabilele individuale;

b) conectivele logice: conjuncţie, negaţie.

Observaţii. *Desigur că se pot utiliza şi alte conective logice cum sînt disjuncţia şi negaţia* $(A \wedge B \to \neg(\neg A \vee \neg B))$, *sau disjuncţie, implicaţie, echivalenţă ce pot fi obţinute din conjuncţie şi negaţie.*

$$A \vee B \Rightarrow \neg(\neg A \wedge \neg B)$$

$$A \to B \Rightarrow \neg A \vee B$$

$$A \equiv B \Rightarrow (A \to B) \wedge (B \to A)$$

Pot fi reduse conectivele logice necesare şi suficiente pentru definirea limbajului de ordin întîi la o singură conectivă "|" definită şi conectiva de incompatibilitate, non-conjuncţia sau functorul Sheffer. O expresie $A|B$ semnifică faptul că A şi B nu sînt ambele adevărate, aşa că se pot transcrie conectivele logice uzuale ca în exemplele de mai jos:

$$\neg A \Rightarrow A|A$$

$$A \vee B \Rightarrow (A|A)|(B|B)$$

$$A \wedge B \Rightarrow (A|B)|(A|B)$$

$$A \to B \Rightarrow A|(B|B)$$

Întrucît reducerea numărului conectivelor logice duce la pierderea semnificaţiei naturale a proceselor inferenţiale, pentru raţiuni practice se acceptă drept primitive toate conectivele logice uzuale ale calculului propoziţional:

c) cuantificatorul existenţial $\exists$. Cuantificatorul universal $\forall$ poate fi exprimat cu ajutorul cuantificatorului existenţial prin transcrierea

$$\forall\, xF \Leftrightarrow \neg(\exists x)\neg F$$

d) simbolurile relaţionale $\theta(i)$, $i \in I$;

e) simbolurile funcţionale $\psi(j)$, $j \in J$;

f) cîte o constantă individuală e_k pentru fiecare $k \in K$.

Se remarcă faptul că unii autori includ printre simbolurile primitve şi pe cel de egalitate. În această situaţie se introduce o regulă suplimentară pentru formarea formelor atomice (adică dacă $E1$ şi $E2$ sînt termeni atunci $E1 = E2$ este o formulă atomică) şi limbajele se numesc "limbaje de ordinul întîi cu egalitate". După alţi autori egalitatea este un simbol relaţional şi deci nu reprezintă o categorie specială a simbolurilor primitive. Alţi autori includ în familia simbolurilor primitive şi clasa simbolurilor delimitatoare formată din parateze, semne de punctuaţie.

Aşa cum cuvintele într-un limbaj nu se aleg la întîmplare, este necesară o mulţime de reguli capabilă să selecţioneze cuvintele limbajului, mulţime numită şi gramatică formală. Formarea termenilor limbajelor de ordinul întîi se poate considera că are loc după o serie de reguli, date mai jos:

- orice variabilă este un termen;
- orice simbol de constantă individuală este un termen;
- dacă $t_1, \ldots t_n$ sînt termeni iar f este un simbol funcţional n-ar atunci $f(t_1, \ldots t_n)$ este un termen;

- orice aplicare a regulilor de mai sus de un număr finit de ori determină un nou termen.

Formulele atomice ale limbajului se obţin prin aplicarea următoarei reguli:
dacă R este un simbol relaţional n-ar şi $t_1, t_2, \ldots t_n$ sînt termeni, atunci $R(t_1, \ldots t_n)$ este o formulă atomică.

Formarea formulelor limbajului $\mathfrak{L}_\sigma$ se realizează după regulile:

- orice formulă atomică este o formulă;
- dacă $F1$, $F2$ sînt formule atunci $\neg F1$, $\neg F2$, $F1 \wedge F2$ sînt formule;
- daca x este o variabilă şi F este formulă, atunci $\exists\, xF$ este o formulă;
- orice aplicare a regulilor de mai sus de un număr finit de ori determină o nouă formulă.

O variabilă este liberă într-o formulă dacă:

- x apare într-o formulă atomică F;
- dacă x este o variabilă liberă în formula F, considerînd şi o altă variabilă y din F îndeplinind condiţia $y \neq x$, atunci x este o variabilă liberă şi pentru formula $\exists\, yF$;
- dacă x este o variabilă liberă în F, atunci x este de asemenea o variabilă liberă şi pentru formulele $\neg F$ şi $F \wedge G$ (în care G este orice formulă).

Se numeşte "expresie bine formată" orice formulă în $\mathfrak{L}_\sigma$. Se numeşte *enunţ* o formulă care nu conţine variabile libere. O variabilă cuantificată universal sau existenţial, este o variabilă legată în formula prefixată de cuantificatorul respectiv.

Dacă A este o formulă şi $x_1, x_2, \ldots x_n$ sînt variabile ce apar în A se poate scrie relaţia dintre numele A al formulei şi variabilele sale sub forma $A(x_1, x_2, \ldots x_n)$. Se defineşte substituţia σ ca o listă finită de perechi cuprinzînd asociaţii ale altor variabile, $\sigma = (x_1 \leftarrow t_1, \ldots x_n \leftarrow t_n)$. Dacă $A(x_1, \ldots x_n)$ este o formulă, aplicarea substituţiei asupra formulei A, notată σA, se realizează prin înlocuirea fiecărei apariţii libere a lui x în A prin t_k pentru $k = \overline{1, n}$

$$\sigma A = A(x_1 \leftarrow t_1, \ldots x_n \leftarrow t_n) = A(t_1, \ldots t_n)$$

Se va nota DOM(σ) pentru o substituţie σ, domeniul substituţiei

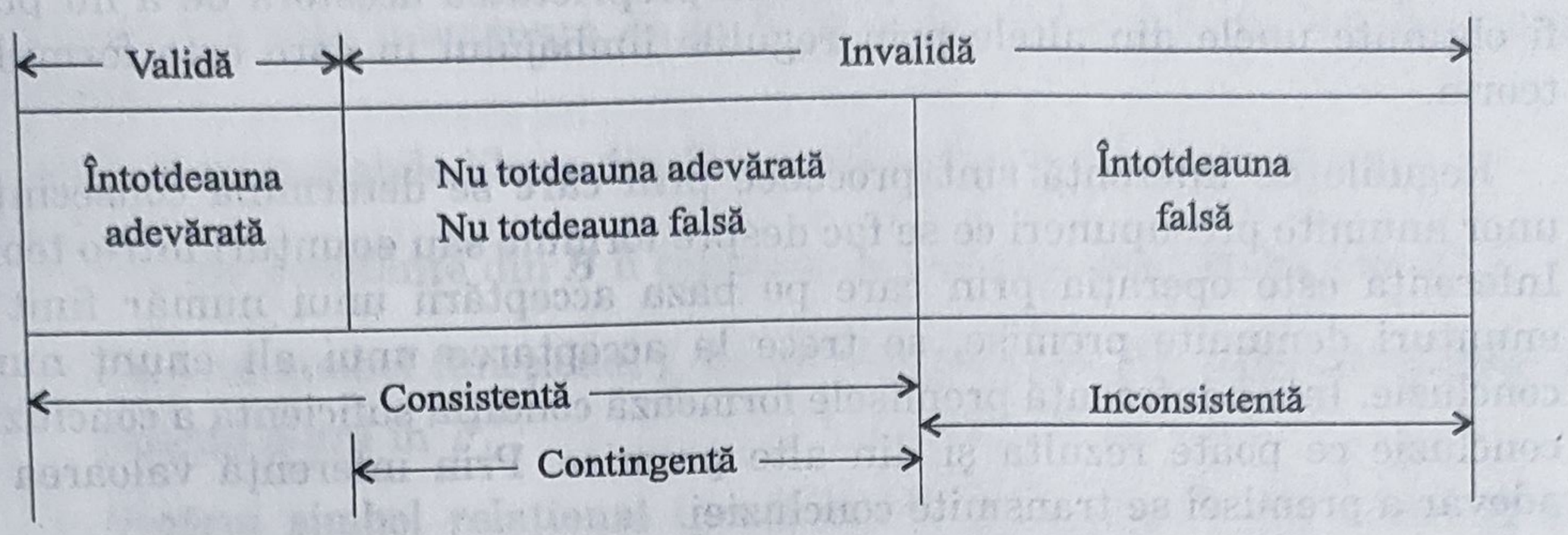
$$\mathrm{DOM}(\sigma) = \{x_i \mid x_i \# \sigma x_i\}$$

şi respectiv

$$\mathrm{REP}(\sigma) = \{t_1 \mid \sigma x_i = t_i\}$$

mulţimea înlocuitorilor substituţiei respective.

▼ ***Fig. 2.1. Relaţia între valoarea de adevăr şi termenii de clasificare a validităţii unei formule***



În semantica oricărui limbaj de ordinul întîi, oricărei formule i se asociază o semnificaţie şi o valoare de adevăr. Mulţimea valorilor de adevăr este specifică limbajului. Pentru logica calculului proporţional şi în calculul cu predicate de ordinul întîi, mulţimea este formată din două valori "adevărat" şi "fals". Se spune că o expresie este validă dacă valoarea sa de adevăr este totdeauna "adevărat", indiferent de valorile de adevăr ale componentelor sale. O expresie este consistentă dacă valoarea de adevăr nu este întotdeauna "falsă". O expresie este invalidă dacă valoarea sa de adevăr nu este întotdeauna "adevărat". Dacă valoarea de adevăr a unei expresii este întotdeauna "fals" se spune că expresia este inconsistentă.

Aşa cum se observă există o categorie de expresii ce nu sînt totdeauna valide şi nici totdeauna inconsistente. Aceste expresii care pentru unele combinaţii ale valorilor de adevăr ale formulelor atomice iau valoarea "adevărat" iar pentru altele iau valoarea "fals", sînt denumite expresii contingente (fig. 2.1).

O mulţime T consistentă de enunţuri într-un limbaj $\mathfrak{L}_\sigma$ este o teorie în acest limbaj. O teorie T_1 este inclusă în altă teorie T_2 ($T_1 \subset T_2$) dacă orice consecinţă logică în teoria T_1 este o consecinţă logică şi în T_2. Două teorii T_1, T_2 sînt echivalente dacă $T_1 \subset T_2$ şi $T_2 \subset T_1$. Dacă pentru orice enunţ S din limbajul $\mathfrak{L}_\sigma$ atît S cît şi $\neg S$ sînt deductibile din T, avem fie $T \vdash S$ fie $T \vdash \neg S$, şi se spune că teoria T este completă. Data fiind o structură α de tip σ, o teorie completă T pe această structură este mulţimea tuturor enunţurilor din $\mathfrak{L}_\sigma$ care sînt valide în α. Limbajul formal de ordinul întii $\mathfrak{L}_\sigma$ în care sînt formulate teoriile despre structurile cu tipul de similaritate σ este denumit limbaj obiect pentru că enunţurile sale sînt obiecte ale acestei teorii. O teorie în care obiectele sînt expresii din $\mathfrak{L}_\sigma$ sau definiţii ale obiectelor sale primitive, sau regulile sale de formare pentru formule şi expresii bine formate se numeşte metateorie, iar limbajul asociat metateoriei, metalimbaj peste $\mathfrak{L}_\sigma$.

Axiomele - sînt formule cu statut de permanentă valabilitate într-o teorie. Multimea axiomelor formează ceea ce se numeşte schema axiomatică.

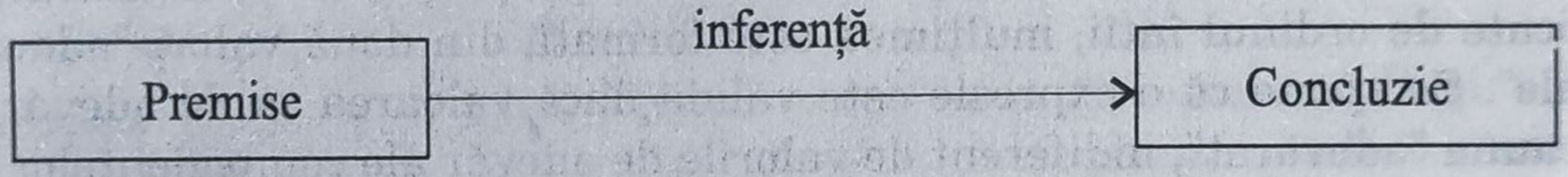
Non-consistenţa sau consistenţa formală a unei teorii presupune lipsa contradicţiilor adică existenţa unei formule şi a negaţiei sale.

Independenţa axiomelor - reprezintă proprietatea acestora de a nu putea fi obţinute unele din altele prin regulile limbajului în care este formulată teoria.

Regulile de inferenţă sînt procedee prin care se determină consecinţele unor anumite presupuneri ce se fac despre formule sau enunţuri într-o teorie. Inferenţa este operaţia prin care pe baza acceptării unui număr finit de enunţuri denumite premise, se trece la acceptarea unui alt enunţ numit concluzie. Într-o inferenţă premisele formează condiţia suficientă a concluziei, concluzie ce poate rezulta şi din alte premise. Prin inferenţă valoarea de adevăr a premisei se transmite concluziei.

O teorie pentru care s-au definit axiome şi reguli de inferenţă este o teorie *axiomatică deductivă*. În general o inferenţă are o structură ca cea din fig. 2.2:

▼ ***Fig. 2.2. Structura unei inferenţe***



Un lanţ inferenţial exprimat în limbaj obiect, ale cărui premise sînt axiome se numeşte *demonstraţie* iar concluzia se numeşte teoremă. Teoria demonstraţiei este formată din metodele prin care regulile de inferenţă sînt aplicate eficient pentru a deduce teoreme din axiome prin "înlănţuire înainte", sau din metodele ce determină validitatea unor teoreme enunţate prin "înlănţuire înapoi". O metodă de bază a inteligenţei artificiale este cea de transformare a enunţurilor în enunţuri a căror validitate este demonstrată.

Altă modalitate de abordare a problemelor în vederea soluţionării ia în consideraţie caracteristicile semantice, exprimate de regula în metalimbaj. In acest scop se concepe o construcţie formală numită model prin care se pun în corespondenţă caracteristicile modelului cu cele ale problemei pe care o modelează. Dacă ansamblul concluziilor rezultate din evaluarea modelului este susceptibil de a fi controlat prin determinări experimentale, calcule sau observaţii, acest ansamblu se constituie în ceea ce se numeşte teoria modelului respectiv. Obiectele teoriei generale a modelelor sînt structurate. Modelele în inteligenţa artificială nu necesită realizări fizice, ele fiind pur conceptuale şi funcţionează pe baza interpretării logice.

Se consideră că limbajele obiect cel mai larg utilizate în inteligenţă artificială sînt limbajul calculului proporţional şi limbajul calculului cu predicate de ordinul întîi. Metalimbajul logicii matematice, în care sînt construite modelele teoretice, cuprinde pe lîngă limbajul obiect şi unele simboluri metalingvistice avînd ca scop asertarea validităţii ($\models$), asertarea deductibilităţii ($\vdash$), valori de adevăr (a-adevărat şi f-fals).

Considerînd o formulă B şi o structură dată:

$$\alpha = <A, R_i^{\alpha}, F_j^{\alpha}, e_k^{\alpha}> \quad i \in I, j \in J, k \in K$$

se spune că φ este o asignare de la B la α dacă următoarele condiţii sînt satisfăcute:

- fiecărei variabile libere din B îi corespunde un element din A;
- fiecărei constante din B îi corespunde un element distinct din e_k^{α};
- fiecărui simbol funcţional f_j de aritate $\psi(s) = n$ din B îi corespunde o funcţie n-ara în F_j^{α};
- fiecărui simbol relaţional (predicat) P_i de aritate $\theta(i) = m$ din B îi corespunde un simbol relaţional (predicat) m-ar în R_i^{α};
- fiecare simbol propoziţional are o valoare de adevăr, a (adevărat) sau f (fals).

O interpretare a formulei B este o pereche ordonată (α, φ), în care α este o structura, iar φ este o asigurare de la B la α. Prin interpretarea formulei B, acesteia i se asociază o semnificaţie în structura α. Unei interpretări (α, φ) a unei formule B i se poate calcula valoarea de adevăr, proces numit şi *valuare* a lui B sub asignarea φ în structura α.

2.2. Limbajul calculului propoziţional

Calculul propoziţional poate fi considerat ca nucleu al oricărui sistem formal. Prima tratare axiomatică a logicii propoziţiilor a fost realizată de matematicianul german Gottlob Frege în lucrarea Begrifschift (1879), însă cu un simbolism greoi ce nu s-a putut impune. Alfred Whitehead şi Bertrand Russel în lucrarea Principia Mathematica (1910-1913) au o contribuţie importantă la desăvîrşirea formulării axiomatico-deductivă a logicii matematice, definind un simbolism mai facil ce este păstrat şi astăzi. Totuşi utilizarea practica este greoaie, notaţia nu păstrează semnificaţia proceselor inferenţiale, sistemul axiomatic nu este independent. Dezvoltările ulterioare făcute de David Hilbert, W. Ackermann şi Paul Bernays fac calculul propoziţional mult mai lucrativ.

Principalele obiective ale introducerii unui sistem formal axiomatic pentru o teorie logica sînt: definirea alfabetului sau a vocabularului, prezentarea regulilor de formare a formulelor sau cuvintelor admise în limbaj, introducerea definiţiilor sau a regulilor de prescurtare, enunţarea axiomelor sau a formulelor admise iniţial, prezentarea regulilor de inferenţă sau a regulilor de derivare a formulelor valide, demonstrarea teoremelor. Cercetările ulterioare conduc la verificarea proprietăţilor metateoretice ale sistemelor axiomatice pentru logica propoziţiilor sau ale altor sisteme logice: nonconcordanţa, completitudinea, independenţa.

Vocabularul sau simbolurile primitive ale sistemului axiomatic Hilbert Ackermann sînt:

$$A = \{a, b, \ldots, x, \neg, \vee, (,)\}$$

în care:

- $a, b, \ldots, x$ sînt variabile individuale (propoziţii) cu două valori de adevăr (adevărat şi fals). Se va considera propoziţia ca o entitate primitivă ce are o singură proprietate aceea de a lua o singură valoare, şi anume valoarea de adevăr;
- variabile propoziţionale formate din negaţie ($\neg$), disjuncţie ($\vee$). Negaţia este o operaţie monadică, disjuncţia o operaţie binară;
- semne de punctuaţie şi grupare.

Deci alfabetul conţine variabile propoziţionale, conective logice şi semne de grupare. Prin definiţie se pot introduce şi ale conective logice. Din considerente de comoditate şi lizibilitate în lucrare se vor utiliza si alte conective logice.

Pentru formarea formulelor în limbajul calculului propoziţional se utilizează reguli de construcţie pornind de la formule elementare ce agregă elementele vocabularului în ansambluri semiotico-sintactice. Se dau mai jos aceste reguli:

- dacă a şi b sînt propoziţii, atunci a; b; $\neg a$; $\neg b$; $a \vee b$; $a \wedge b$; $a \rightarrow b$; sînt formule atomice ale calculului propoziţional;
- orice formulă atomică este o formulă;
- dacă $F1$ şi $F2$ sînt formule atunci $\neg F1$; $\neg F2$; $F1 \vee F2$; $F1 \wedge F2$; $F1 \rightarrow F2$; $F1 \equiv F2$ sînt formule;
- orice formulă inclusă între paranteze este tot o formulă.

Se dau mai jos cîteva dintre axiomele calculului propoziţional:

$\vDash A \vee A \rightarrow A$	tautologie
$\vDash A \rightarrow A \vee B$	adiţiune
$A \vee B \rightarrow B \vee A$	comutativitate
$(A \rightarrow B) \rightarrow ((C \vee A) \rightarrow (C \vee B))$	principiul insumării

Din combinaţii sintactice valide de propoziţii se pot alcătui trei tipuri de formule:

- *tautologii* - formule ce au tot timpul valoarea "adevărat" indiferent de valoarea de adevăr a propoziţiilor componente;
- *contradicţii* - formule ce au totdeauna valoarea "fals" indiferent de valoarea de adevăr a propoziţiilor componente;
- *formule variabile* - ale căror valori de adevăr sînt dependente de valoarea de adevăr a propoziţiilor componente.

Pentru judecăţi şi aprecieri asupra formulelor scrise în limbajul calculului propoziţional se folosesc semne metalingvistice:

- $\vDash A$ - indică faptul că A este validă, adică în tabela sa de adevăr se află numai valoarea a, indiferent de valorile de adevăr ale propoziţiilor atomice componente;
- $A \vDash B$ indică faptul că B este o consecinţă validă a formulei A şi deci B are valoarea "adevărat" peste tot unde A are valoarea "adevărat";
- $\vdash A$ semnifică faptul că A este deductibilă din axiome sau alte formule prin aplicarea regulii de inferenţă *"modus ponens"*;
- $A \vdash B$ - formula B este deductibilă din formula A, adică B se obţine ca o secvenţă finită de formule între care se găseşte şi A.

Pentru a se obţine consecintele asignării unui set de valori de adevăr pentru variabilele propoziţionale $A, B, C,$ variabilele ce sînt componente elementare ale unui model exprimat prin formula $(A \vee B) \rightarrow (B \wedge C)$ se determină semnificaţia fiecărui grup distinct asociat variabilelor A, B, C în modelul teoretic.

Se exemplifică mai jos considerînd că valorile de adevăr pentru A, B, C sînt f, a, f.

▼ ***Fig. 2.3 Determinarea semnificaţiei***

$$\frac{(A \vee B) \rightarrow (B \wedge C)}{(f \vee a) \rightarrow (a \wedge f)} \qquad \frac{a \rightarrow f}{f}$$

Rezultatul obţinut este determinat de aplicarea în mod repetat a unei reguli de substituire ce rezultă din teorema:

Fie F o formulă care conţine numai propoziţiile $P_1, \ldots P_n$ şi fie $A_1, \ldots, A_n$ formule în calculul propoziţional. Fie F' obţinut din F prin substituirea lui $P_1, \ldots P_n$ simultan cu formulele $A_1, \ldots A_n$. Dacă $\vDash F$, rezultă de asemenea $\vDash F'$. Cu alte cuvinte substituţia conservă adevărul.

Prin teorema de mai sus se pot determina formule valide (tautologii) de importanţă mare pentru calculul propoziţional. Aceste tautologii pot fi considerate drept o schemă axiomatică a unui sistem de deducţie naturală bazat pe calculul propoziţional. Pentru un sistem de deducţie naturală este caracteristic faptul că oferă pentru fiecare conectivă axiome din care se deduc reguli de introducere şi reguli de eliminare, ca sistem de reguli eficace în IA:

- *regula de* $\rightarrow$ *introducere* arată că dacă $\Gamma, A \vdash B$, atunci $\Gamma \vdash A \rightarrow B$;
- *regula de* $\rightarrow$ *eliminare* arată că dacă A este adevărat şi $A \rightarrow B$ este adevarat, atunci se poate afirma că B este adevărat

$$A, A \rightarrow B \vdash B$$

(regula este cunoscută şi sub denumirea de "modus ponens")

- *regula de* $\wedge$ *- introducere* arată că dacă A este adevărat şi B este adevărat, atunci $A \wedge B$ este adevărat, adică $A, B \vdash A \wedge B$;

♦ *regula de* $\wedge$ - *eliminare* arată că dacă $A \wedge B$ este adevărat, se afirma că A este adevărat şi B este adevărat, adică:

$$A \wedge B \vdash A$$
$$A \wedge B \vdash B$$

♦ *regula de* $\vee$ - *introducere* arată că dacă A este adevărat, atunci se poate afirma că pentru orice B avem următoarele formule adevărate:

$$A \vdash A \vee B$$
$$A \vdash B \vee A \text{ (conform comutativităţii);}$$

♦ *regula de* $\vee$ - *eliminare* arată ca dacă C se deduce dintr-un lanţ inferenţial $(\Gamma, A \vee B)$ a cărui ultimă formulă este $A \vee B$, atunci C se poate deduce atît dintr-un lanţ inferenţial (Γ, A) cît şi dintr-un lanţ inferenţial (Γ, B)

$$(\Gamma, A \vee B \vdash C) \rightarrow (\Gamma, A) \vdash C$$
$$(\Gamma, A \vee B \vdash C) \rightarrow (\Gamma, B) \vdash C$$

♦ *regula de* $\neg$ - *introducere,* arată că dacă dintr-un lanţ inferenţial (Γ, A) se deduce atît B cît şi negaţia lui, respectiv $\neg B$, atunci se poate deduce şi negaţia lui A, deci din $(\Gamma, A) \vdash B$ şi $(\Gamma, A) \vdash \neg B$ atunci $\Gamma \vdash \neg A$ (inferenţa cunoscută sub numele de reducere la absurd);

♦ *regula de* $\neg$ - *eliminare,* arată că dacă se neagă negaţia lui A atunci se afirmă A. Deci, $\neg \neg A \vdash A$ (principiul dublei negaţii);

♦ *regula de* $\equiv$ *introducere* arată că dacă este adevărat ca $A \rightarrow B$ şi este adevărat că $B \rightarrow A$ atunci A şi B sînt echivalente

$$(A \rightarrow B, B \rightarrow A) \vdash (A \equiv B)$$

♦ *regula de* $\equiv$ - *eliminare,* arată că dacă A şi B sînt echivalente, sînt simultan adevărate şi implicaţiile $A \rightarrow B$ şi $B \rightarrow A$

$$A \equiv B \vdash (A \rightarrow B)$$
$$A \equiv B \vdash (B \rightarrow A)$$

Deducerea valorii de adevăr a oricărei formule exprimată în limbajul calculului propoziţional se poate realiza prin aplicarea de un număr finit de ori a regulilor de inferenţă. Se spune că o formulă B este o *deducţie formală* din formulele $A_1, A_2, \ldots A_n$ $(A_1, A_2, \ldots A_n \vdash B)$ dacă B este fie una din formulele $A_1, A_2, \ldots A_n$ sau una din axiomele sistemului formal, sau provine din două formule $A_1, \ldots A_n$ prin aplicarea regulii de inferenţă modus ponens.

Se spune că B este o *consecinţă validă* a lui $A_1, A_2, \ldots A_n$ dacă prin construirea model teoretică a tabelei de adevăr pentru $A_1, \ldots A_n \vdash B$ în funcţie de valorile de adevăr $P_1, \ldots P_m$ se obţine valoarea "a" pentru B în toate cazurile unde $A_1, \ldots A_m$ sînt simultan adevărate.

Teoremă. $A_1, A_2, \ldots A_m \vdash B$ dacă şi numai dacă $A_1, A_2, \ldots A_{m-1} \vdash A_m \rightarrow B$. Se obţine astfel următorul corolar: $A_1, A_2, \ldots A_m \vdash B$ dacă şi numai dacă $\vdash A_1 \rightarrow (\ldots(A_{m-1} \rightarrow (A_m \rightarrow B))\ldots)$.

Demonstraţia teoremei se face pornind de la analiza tabelelor de adevăr pentru $A_1,... A_m$, B şi $A \to B$, în care se substituie A_m lui A apoi se selectează numai acele linii în care $A_1,... A_{m-1}$ sînt simultan adevărate.

Teorema deducţiei. Dacă $A_1,... A_{m-1}, A_m \vdash B$ atunci $A_1,..., A_{m-1} \vdash A_m \to B$.

Se obţine din această teoremă corolarul. Dacă $A_1,..., A_{m-1}, A_m \vdash B$ atunci $\vdash A_1 \to (...(A_{m-1} \to (A_m \to B)...)$ ce sînt rezultate de mare importanţă pentru teoria deducţiei.

2.3. Limbajul calculului cu predicate de ordinul întîi

Termenul de predicat a fost utilizat pentru prima dată de Aristotel în teoria sa silogistică. În viziunea sa predicatul era un termen al unei judecăţi şi anume cel care descrie ce se afirmă despre subiect. În sens aristotelic o judecată are forma "Socrate e muritor", în care "Socrate" şi "muritor" sînt predicate. În logica matematică termenul de predicat este folosit în accepţiunea de formulă deschisă sau funcţie logică în forma muritor (x), tata (x), fiu(x, y), etc. Formulele de acest tip pot deveni enunţuri sau propoziţii dacă variabilele iau ca valori constante individuale sau sînt cuantificate universal ori existenţial. Logica predicatelor aşa cum i-a fost atribuită lui G. Frege a avut scopul de creare a unui limbaj capabil să descrie enunţurile matematice şi să garanteze corectitudinea raţionamentelor. Limbajul logicii predicatelor dispune de instrumente pentru a defini tipuri de obiecte matematice cum sînt: numere, variabile numerice, funcţii şi operaţii matematice, relaţii, propoziţii şi teoreme matematice. În logica predicatelor s-au standardizat schemele de inferenţă ce asigură trecerea de la enunţuri adevărate sau premise la alte enunţuri adevărate numite şi consecinţe sau concluzii.

În mod normal oamenii nu gîndesc în limbajul teoriilor logice, ci în limbile naturale, în raport cu scopurile pe care le urmăresc. Este greu în a garanta corectitudinea raţionamentelor efectuate, fiind dependent de opinia celui ce evaluează raţionamentul. A testa validitatea unui astfel de raţionament este o chestiune dificilă, chiar dacă logica matematică modernă dispune de mijloace adecvate, întrucît este mai întîi necesară transpunerea limbajului natural într-un formalism logico-formal.

Fiecare teorie logică are particularităţi în ceea ce priveşte semnele primitive (alfabetul), operaţiile logice admise, numărul şi natura valorilor de adevăr, principiile şi metodele prin care se leagă formulele rezultate cu elementele lumii reale. În acest paragraf se urmăreşte prezentarea principalelor componente ale limbajului calculului cu predicate, semantica logicii predicatelor, teoria inferenţelor logice.

În principal se poate afirma ca limbajul calculului cu predicate de ordinul întîi reprezintă o extensie a calculului propoziţional prin luarea în consideraţie a componentelor propoziţiilor, pentru a interpreta modul în care acestea

participă la determinarea valorilor de adevăr ale formulelor. Extensia se referă la simbolurile primitive ale limbajului. Propoziţia este considerată a avea o structură cu două componente:

- *predicatul* - ce specifică semnificaţia funcţională a propoziţiei;
- *argumentul predicatului* - specifică semnificaţia suportului funcţional al propoziţiei.

Orice predicat $P(x)$ joacă rolul de funcţie propoziţională deoarece pentru orice valoare a argumentului independent x, acesta ia ca valoare o propoziţie, cu valoarea adevărat sau fals. Numărul de obiecte individuale ce compun argumentul reprezintă aritatea predicatului. Astfel pentru:

$n = 0$ - predicatul este o propoziţie;
$n = 1$ - predicatul specifică o proprietate a argumentului;
$n = 2$ - predicatul specifică o proprietate binară;
$n = 3$ - relaţie n-ară.

Orice predicat este de ordinul întîi dacă variabilele sale nu sînt predicate (se poate spune despre astfel de variabile că pot fi considerate predicate de ordinul zero). Un predicat care are cel puţin una din variabiele sale individuale predicate de ordinul întîi se spune că este un predicat de ordinul doi. Se poate spune în general că un predicat este de ordin n dacă cel puţin una din variabilele sale este un predicat de ordinul n - 1. Pentru calculul cu predicate de ordinul întîi formulele se formează după regulile limbajului calculului propoziţional, în care variabilele propoziţionale sînt înlocuite cu predicate. În relaţia folosită pentru predicate, variabilele au semnificaţie nu prin simbolurile folosite, ci prin poziţia şi rolul pe care îl joacă în definirea predicatului respectiv. Variabilele folosite în aceste notaţii sînt denumite variabile formale, iar notaţia definitorie pentru predicate este denumită *expresie predicativă primară sau ion.*

Se spune că $P(x, y)$ şi $P(u, v)$ sînt diferite ataşări de variabile formale pentru aceeaşi expresie predicativă primară $P(\_1, \_2)$, iar $P(a, b)$ este o instanţă a acestei expresii predicative în care a şi b au fost asignate primului şi respectiv celui de-al doilea argument al predicatului. Pentru aprecierea valorii de adevăr în raport cu domeniul în care iau valori variabilele individuale ale predicatelor componente, se introduc o serie de operatori unari denumiţi şi cuantificatori ce acţionează asupra variabilelor.

- cuantificatorul universal indică faptul că formula prefixată este validă pentru toate valorile posibile în domeniul variabilei x.
- cuantificatorul existenţial indică faptul că formula prefixată este validă pentru cel puţin o valoare din domeniul variabilei x.

Orice formulă bine formată prefixată cu un cuantificator ($\exists$ sau $\forall$) aplicat unei variabile individuale care apare în formulă, este o formulă bine formulată. O formula F prefixată de cuantificatori universali sau existenţiali, cum sînt $\exists xF$ şi $\forall xF$ este denumită *scop* al cuantificatorului respectiv. Rezultă deci, că în limbajul calculului cu predicate de ordinul întîi orice formulă bine formată, prefixată cu un cuantificator ($\exists$ sau $\forall$) aplicat unei variabile individuale care apare în formulă, este o formulă bine formulată.

Teoria demonstraţiei la calculul cu predicate de ordinul întîi foloseşte pe lîngă axiomele şi regulile de inferenţă ale limbajului calculului propoziţional şi următoarele axiome:

$\vdash \forall\, xA(x) \to A(r)$

$\vdash A(r) \to \exists\, xA(x)$ în care:

x este orice variabilă

$A(x)$ este o formulă

r - orice variabilă nu neapărat distinctă de x

$A(r)$ - rezultatul substituţiei libere cu r a oricărei apariţii libere a variabilei x în $A(x)$.

Reguli de inferenţă pentru calculul cu predicate de ordinul întîi:

($\forall$) regula: dacă $\vdash C \to A\,(x)$ atunci $\vdash C \to \forall\, xA(x)$

($\exists$) regula: dacă $\vdash A(x) \to C$ atunci $\vdash \exists\, xA(x) \to C$

în care: x - orice variabilă; $A(x)$ o formulă; C este o formulă ce nu conţine variabila liberă x.

În general se spune că substituţia lui x cu r în $A(x)$ este liberă, dacă pentru fiecare apariţie liberă a lui x în $A(x)$ rezultă o apariţie liberă corespunzătoare a lui r în $A(r)$.

Un predicat într-o formulă depinde de atîtea variabile cîte litere apar libere în el. Ca exemplu în formula:

$$\forall\, y \exists\, w\, P(y, w, z) \to \forall\, x \exists\, w\, P(x, w, z)$$

predicatul 3-ar P depinde de o singură variabilă în ambele sale apariţii deoarece la prima apariţie variabilele y şi w sînt legate, singura variabilă liberă fiind z iar la a doua apariţie variabilele x şi w sînt legate singura variabilă liberă fiind tot z.

Orice formulă ce nu conţine apariţii libere ale variabilelor sale, se numeşte formulă inclusă. Cînd includerea unei formule este dată numai de unele din variabilele sale este recunoscută ca închiderea la variabilele respective. Dacă unei formule i se adaugă cuantificatorul universal pentru fiecare variabilă liberă, devine închiderea universală a formulei considerate $\mathcal{F}$. O formulă $\mathcal{F}$ este o *consecinţă validă* a formulelor $A_1, \ldots A_m$ relativ la variabilele libere $x_1, x_2, \ldots x_r$ şi se notează:

$$A_1, \ldots, A_m \vDash x_1, \ldots, x_r\, \mathcal{F}$$

dacă se consideră constante toate celelalte variabile, cu excepţia celor specificate. Similar, o formulă este *deductibilă* din $A_1, \ldots, A_m$ relativ la variabilele $x_1, \ldots x_r$ şi se notează

$$A_1, \ldots, A_m \vdash x_1, \ldots, x_r\, \mathcal{F}$$

dacă se consideră deductibilitatea din formulele $A_1, A_2, \ldots, A_m$ cu menţinerea constantă a tuturor celorlalte variabile.

Calculul cu predicate de ordinul întîi se bazează pe rezultatele unor teoreme importante cum sînt:

- teorema de substituire pentru expresii predicative primare;
- teorema de substituire pentru variabile individuale;
- teoreme de ∀ - introducere, ∀ - eliminare, ∃ - introducere, ∃ - eliminare.

2.4. Teoria inferenţelor logice

A decide asupra unei formule înseamnă a determina printr-o metodă adecvată dacă formula este validă sau infirmată. A. Chaurk (1936) demonstrează faptul că logica predicatelor nu este o teorie complet decidabilă, deci nu există o metodă universală prin care într-un număr finit de paşi să decidă dacă o formulă bine formată este validă sau infirmată. Se poate afirma însă, că există diferite clase de formule cum sînt forma normală prenex, forma normală Skolem ce pot fi descrise într-un număr finit de paşi. Metode cum sînt metodele tablourilor semantice sau metoda arborilor de decizie nu vor fi prezentate aici, atenţie deosebită fiind acordată metodei rezoluţiei.

Modul de desfăşurare a raţionamentelor pentru obţinerea de concluzii valabile din premisele enunţate în probleme, este obiectivul oricărui sistem inteligent de rezolvare a problemelor. Raţionamentele efectuate asupra formulelor exprimate în limbajul logicii matematice, cuprind o serie de raţionamente elementare specifice numite reguli de inferenţă. Din analiza limbajului calculului cu predicate de ordinul întîi şi a enunţurilor (axiome şi reguli) exprimate în acest limbaj se defineşte o teorie a calculului cu predicate de ordinul întîi notată T. Se observă că enunţurile sînt de 3 feluri:

- enunţuri referitoare la propoziţie, care alcătuiesc o teorie a propoziţiilor $T1$;
- enunţuri referitoare la predicate, care alcătuiesc o teorie a predicatelor $T2$;
- reguli de inferenţă ce alcătuiesc o teorie a demonstraţiei $T3$.

Există evident relaţii $T1 \subset T$, $T2 \subset T$, $T3 \subset T$ şi $T = T1 \cup T2 \cup T3$.

Teoria inferenţelor logice este formată din abstracţia tuturor raţionamentelor elementare a căror valabilitate nu depinde de obiectele implicate în raţionament ci doar de structura premiselor. Gerhard Gentpen face prima abstractizare ştiinţifică ce permite o definire sistematică a regulilor de inferenţă, formalizînd schemele de derivare pe baza cărora se pot construi mecanismele inferenţiale în sistemele cognitive şi rezolutive. Dacă s-ar considera că un sistem de reguli de inferenţă este consistent cu sistemul axiomatic al unei teorii şi deci nu ar mai fi necesar, ar rezulta mari dificultăţi în utilizarea instrumentelor logicii matematice, deoarece derivarea formulelor

într-un sistem axiomatic ce foloseşte ca singură inferenţă substituţia determină calcule laborioase. O astfel de abordare va atrage după sine imposibilitatea folosirii lanţului de transformări operate drept justificare a unui raţionament. Semnificaţia operaţională este dată de faptul că au un conţinut semantic. Este motivul pentru care în IA sistemele axiomatic şi inferenţial trebuiesc să coexiste. Oricum, principiul rezoluţiei este o regulă de inferenţă mai generală decît modus ponens, modus tollens, principiul raţionamentului ipotetic sau legea tranzitivităţii. Aplicarea metodei necesită mai întîi aducerea la o formă standard numită formă clauzală.

Ca urmare cea mai obişnuită modalitate de reprezentare a regulilor de inferenţă foloseşte forma clauzală a expresiilor în limbaje de ordinul întîi. Iată cîteva tipuri de clauze:

(i) Dacă $A_1 \wedge A_2 \wedge ... \wedge A_m$, atunci B

(ii) B, dacă $A_1 \wedge A_2 \wedge ... \wedge A_m$

Ambele forme descriu aceeaşi clauză avînd o consecinţă unică şi un antecedent complex.

(iii) Dacă $A_1 \wedge A_2 \wedge ... \wedge A_m$, atunci $B_1 \vee B_2 \vee ... \vee B_n$

(iv) $B_1 \vee B_2 \vee ... \vee B_n$, dacă $A_1 \wedge A_2 \wedge ... \wedge A_m$

(v) $A_1, A_2,..., A_m \rightarrow B_1, B_2,..., B_n$

(vi) $B_1, B_2,..., B_n \leftarrow A_1, A_2,..., A_m$.

În (iii) - (vi) au fost prezentate în dublă scriere clauze cu consecinţe alternative. În (v) şi (vi) s-au omis conectivele de conjuncţie la premisă şi disjuncţie la concluzie.

Forma generală a unei clauze complexe este dată în (vii)

(vii) Pentru orice $x_1,..., x_k B_1, B_2,..., B_n \leftarrow A_1, A_2,..., A_m$ unde $x_1,..., x_k$ sînt variabile individuale, $B_1, B_2,..., B_n$ sînt concluzii alternative, $A_1, A_2,..., A_m$ sînt condiţii valabile simultan.

Se va defini un literal ca o formulă atomică sau negaţia ei. Se numesc literali pozitivi formulele atomice şi literali negativi negaţia acestora. Cu aceste precizări o clauză în logica predicatelor, este un set de literali conectaţi prin disjuncţie. De regulă în scrierea clauzei se omite semnul disjuncţiei aceasta apărînd ca mulţimi de literali. Tipologia clauzelor este şi ea variată. Pot exista clauze alcătuite dintr-un singur termen numite şi clauze unitare, clauze cu n literali, clauze negative, clauze Horn. Se numesc clauze Horn acele clauze ce conţin cel mult un literal pozitiv. Se poate uşor observa că o clauză Horn provine dintr-o implicaţie de forma

$$A_1, A_2,..., A_m \vdash B$$

unde $A_1,..., A_m$ sînt condiţiile şi B consecinţa.

Cu aceste precizări este important a arăta că orice formulă bine formată şi închisă în logica predicatelor poate fi adusă la forma clauzală. Pentru a se aduce o formulă din logica predicatelor de ordinul întîi la forma clauzală este necesară parcurgerea următoarelor etape:

(i) eliminarea implicaţiilor şi echivalentelor;

(ii) coborîrea implicaţiilor pe semnele predicative;

(iii) standardizarea variabilelor;

(iv) eliminarea cuantificatorilor existenţiali:
(v) eliminarea cuantificatorilor universali;
(vi) reducerea expresiei la conjuncţii de clauze disjunctive;
(vii) eliminarea operatorilor propoziţionali;
(viii) redenumirea variabilelor.

Eliminarea implicaţiilor şi echivalentelor are loc pe baza relaţiilor de mai jos

$A \supset B \equiv \neg A \vee B$

$A \subset B \equiv A \vee \neg B$

$\equiv (A \equiv B) \equiv (\neg A \vee B) \wedge (A \vee \neg B).$

Al doilea pas se bazează pe legile negaţiei

$\neg\neg A \equiv A$

$\neg(A \wedge B) \equiv \neg A \vee \neg B$

$\neg(A \vee B) \equiv \neg A \wedge \neg B$

$\neg(\forall x)\, A(x) \equiv \exists x\, \neg A(x)$

$\neg(\exists x)\, A(x) \equiv \forall x\, \neg A(x)$

Pasul de standardizare a variabilelor cere ca o variabilă într-o formulă să fie cuantificată într-un singur fel. În acest context nu se permite ca o variabilă într-o formulă să fie cuantificată atît existenţial cît şi universal ca în $\exists y\, Py \wedge \forall y\, Qy$. Formula se va rescrie $\exists y\, Py \wedge \forall z\, Qz$, cu z o variabilă individuală oarecare, diferită de y.

Pasul privind eliminarea cuantificatorului existenţial se sprijină pe teorema Skolem şi pot fi utilizate următoarele reguli:

1) Dacă un cuantificator existenţial nu este precedat în formula dată de nici un cuantificator universal se va elimina variabila de sub cuantificatorul existenţial şi în formulă va fi înlocuită cu o constantă ce nu mai apare. Spre exemplificare se consideră formula $\exists x\, (P(x) \wedge R(x))$ în care variabila prefixată este x şi nu este precedată de cuantificatori universali. Conform strategiei prefixul $\exists x$ va fi eliminat şi o constantă c, nouă, va fi dispusă în locul aparıţiilor lui x, obţinînd $P(c) \wedge R(c)$.

2) Dacă cuantificatorul existenţial este precedat în prefix de cuantificatori universali se va elimina cuantificatorul existenţial din prefix şi orice apariţie în formulă a variabilei prefixate va fi substiuită printr-un semn funcţional ce depinde de toate variabilele de sub cuantificatorii universali ce preced cuantificatorul existenţial în discuţie. Dacă se consideră formula $\forall x \forall y \exists z\ (P(x, y, z) \wedge R(x, z))$ se observă că $\exists z$ este precedat de cuantificatorii universali $\forall x$ şi $\forall y$. Urmînd regula de modificare se va elimina din prefix z iar z se va înlocui prin $f(x, y)$, obţinînd $\forall x \forall y\ (P(x, y, f(x, y)) \wedge R(x, f(x, y)))$.

Cuantificatorii universali rămaşi în formule vor fi eliminaţi conform convenţiilor de rescriere şi interpretare semantică, iar celelalte etape sînt destul de simple.

Se obţine deci pentru forma generală a regulilor de inferenţă expresia:

$$\frac{A_1 \dots A_m}{B_1 \dots B_n} \qquad (m > 1, n > 1)$$

în care $A_1, A_2, \dots, A_m$ sînt clauzele ce alcătuiesc premisa regulii, iar $B_1, B_2, \dots, B_n$ sînt clauzele ce alcătuiesc concluzia. Pornind de la această regulă generală o regulă de inferenţă este o clauză de clauze. O clauză se interpretează ca o concluzie alternativă, condiţionată de valabilitatea simultană a formulelor ce alcătuiesc premisa.

Cazuri extreme:

- $m = 0 \vdash B_1, B_2, \dots, B_r$ adică în mod necondiţionat este adevărat $B_1, B_2, \dots, B_r$ adică avem o aserţiune;
- $n = 0$ $A_1, A_2, \dots, A_m \vdash$ indicînd faptul că din $A_1 A_2 \dots A_m$ se deduce falsul întrucît nu se deduce nimic; deci se obţine o negare a premisei;
- $n = m = 0$ se obţine clauza vidă $\varnothing$ adică o propoziţie falsă.

Pe baza celor de mai sus se pot clasifica regulile de inferenţă în:

a) *reguli de inferenţă structurală* - care se referă la structura clauzelor, indiferent de natura conectivelor ce apar în formulele componente;

b) *reguli de inferenţă logică* - reguli specifice pentru introducerea sau eliminarea conectivelor logice.

Se numeşte regulă de demonstrare o succesiune de reguli de inferenţă ce statisfac condiţiile:

- formulele ce apar în concluzia unei reguli figurează în cel mult o premisă a altei reguli;
- fiecare formulă exceptînd eventual o formulă finală apare în premisa cel puţin a unei reguli;
- regulile în ansamblu nu conţin circularităţi, adică formulele nu generează cicluri.

Acele formule care nu apar în concluzia unei reguli de inferenţă se numesc formule iniţiale.

Un *fir demonstrativ* este format dintr-o succesiune de formule componente ale regulii de demonstrare ce satisfac condiţiile:

- prima formulă a succesiunii este o formulă iniţială;
- ultima formulă a succesiunii este o formulă finală;
- fiecare premisă (cu excepţia ultimei) este o premisă pentru acea regulă de inferenţă a cărei concluzie este chiar premisa regulii următoare.

2.4.1. Reguli de infereţă structurală

Dacă A, B, C, D sînt formule oarecare şi Γ, Δ, θ, ψ, şiruri de formule oarecare separate prin virgulă, conectiva implicită între formulele individuale este cea care corespunde poziţiei pe care acestea le ocupă în clauză, (conjuncţia pentru antecedent şi disjuncţia pentru succedent). Se pot stabili următoarele reguli de inferenţă structurală:

Regula de rafinare

- la antecedent $\dfrac{\Gamma \vdash \psi}{A, \Gamma \vdash \psi}$
- la succedent $\dfrac{\Gamma \vdash \psi}{\Gamma \vdash \psi, A}$

Regula de simplificare

- la antecendent $\dfrac{A, A, \Gamma \vdash \psi}{A, \Gamma \vdash \psi}$
- la succedent $\dfrac{\Gamma \vdash \psi, A, A}{\Gamma \vdash \psi, A}$

Regula de intervertire

- la antecedent $\dfrac{\Gamma, B, C, \Delta \rightarrow \psi}{\Gamma, C, B, \Delta \rightarrow \psi}$
- la succedent $\dfrac{\Gamma \vdash \psi, B, C, \Delta}{\Gamma \vdash \psi, C, B, \Delta}$.

Regula de eliminare a secţionării

- $D, \Delta \vdash \wedge$
- $\dfrac{\Gamma \vdash \psi, D}{\Gamma, \Delta \vdash \wedge, \psi}$.

2.4.2. Reguli de inferenţă logică

Varietatea acestor reguli este dată de tipul conectivelor logice, de modul în care conectiva logică apare în formule, de poziţia din clauza afectată prin inferenţă şi de tipul formulelor componente ale regulii. În funcţie de conectivele logice există reguli de inferenţă pentru conjuncţie $\wedge$, disjuncţie $\vee$, implicaţie, echivalenţă, negaţie, pentru $\exists$, respectiv $\forall$.

Modul în care conectiva apare în inferenţă face separarea între regulile ce introduc conective şi regulile de eliminare a conectivelor. Funcţie de poziţia din clauza afectată avem regulă ce operează la antecedent respectiv la succedent.

După tipul formulei avem:

- reguli care se aplică la componentele unei formule oarecare;
- reguli care se aplică în cazul în care componentele sînt reprezentate în forma clauzală.

Dintre aceste reguli se pot remarca:

- regulile de introducere a conjuncţiei;
- regulile de eliminare a conjuncţiei;
- regulile de introducere a disjuncţiei;
- regulile de eliminare a disjuncţiei;
- regulile de introducere a implicaţiei;
- regulile de eliminare a implicaţiei;
- regulile de introducere a echivalenţei;
- regulile de eliminare a echivalenţei;
- regulile de introducere a negaţiei;
- regulile de eliminare a negaţiei;
- regulile de introducere a cuantificatorilor;
- regulile de eliminare a cuantificatorilor.

2.5. Tehnici fuzzy

2.5.1. Teoria seturilor fuzzy

Pentru facilitarea introducerii în teoria seturilor fuzzy se porneşte de la o scurtă discuţie a setului teoretic al funcţiilor de apartenenţă. În acest scop, se porneşte de la existenţa setului universal X la care alte mulţimi sînt subseturi, ca de exemplu pentru setul $A(A \subset X)$, se spune că $a \in A$ implică $a \in X$. Date fiind postulatele setului universal, orice set A poate fi reprezentat prin utilizarea funcţiei de apartenenţă, numită şi altfel funcţie de adevăr, funcţie de evaluare sau funcţie caracteristică. Funcţia de apartenenţă $u_A(x)$ a setului A este definită relativ la setul universal X prin:

$$u_A(x) = \begin{cases} 1 & \text{dacă } x \in A \\ 0 & \text{dacă } x \notin A \end{cases}.$$

În consecinţă, orice subset A al lui X poate fi reprezentat în forma $A = \{x \mid u_A(x)\}$.

Specific, fiecare x din X este asociat cu funcţia sa de evaluare. Dacă A este finită, poate fi reprezentată ca o listă a tuturor elementelor lui X, fiecare număr urmat de un slash după care, apare fie valoarea 1, fie valoarea 0 după cum elementul aparţine lui A sau nu aparţine lui A.

Exemplu: Considerînd setul universal $X = \{1, 2, 3, 4\}$, dacă $A = \{2, 4\}$ utilizarea funcţiei de apartenenţă va da $A = \{1/0, 2/1, 3/0, 4/1\}$.

Astfel spus într-o reprezentare simbolică fiecare simbol din X are ataşat un "flag", dacă indica 1 acesta aparţine setului A, iar dacă indică 0 nu aparţine setului A.

Este necesară discutarea setului teoretic al operaţiilor referitoare la caracterizarea funcţiei de apartenenţă. Considerînd două serturi A şi B reuniunea $(A \cup B)$ conţine toate elementele ce aparţin fie lui A fie lui B. Utilizarea reprezentării standard va duce la

$$A \cup B = \{x \mid x \in A \text{ sau } x \in B\}$$

pe cînd prin funcţia de apartenenţă

$$A \cup B = \{x \mid u_{A \cup B}(x) = \max(u_A(x), u_B(x))\}.$$

Cu X şi A din exemplul de mai sus şi cu

$$B = \{1/1, 2/1, 3/0, 4/0\} = \{1, 2\}$$

se obţine:

$$A \cup B = \{1/1, 2/1, 3/0, 4/1\} = \{1, 2, 4\}.$$

Intersecţia dintre seturile A şi B, $A \cap B$ este reprezentată de setul elementelor ce aparţin atît lui A cît şi lui B, adică:

$$A \cap B = \{x \mid u_{A \cap B}(x), \text{ unde } u_{A \cap B}(x) = \min(u_A(x), u_B(x))\}.$$

Diferenţa între A şi B va fi reprezentată ca

$$A - B = \{x \mid u_{A-B}(x), \text{ unde } u_{A-B}(x) = \min(u_A(x), 1 - u_B(x))\}.$$

În acest caz atunci cînd A este egal cu X, B reprezintă complementarea lui X, notată şi

$$B^c = X - A = \{x \mid u_B(x), \text{ unde } u_B^c(x) = 1 - u_B(x))\}$$

deci

$$\min(u_X(x), 1 - u_B(x)) = \min(1, 1 - u_B(x)) = 1 - u_B(x)$$

Întrucît operaţiile de intersecţie şi reuniune sînt comutative şi asociative, operaţia poate fi generalizată pentru orice subset al claselor arbitrare şi formulele de Morgan sînt valabile.

Prin setul teoretic, se arată dacă un element aparţine sau nu unui set. Ca exemplu, dacă setul universal X este reprezentat de mulţimea tuturor poligoanelor, setul triunghiurilor este definit prin

$A = \{x. \mid x$ are trei laturi$\}$ este un subset al lui X.

Oricum pentru un particular $x \in X$ sînt posibile două situaţii exclusive: (1) $x \in A$ în care caz $u_A(x) = 1$, sau (2) $x \notin A$, caz în care $u_A(x) = 0$. Însă excluderea mutuală nu poate fi strict aplicabilă în orice tip de raţionare. Dacă se consideră setul animalelor X cu toate speciile sale nu pot fi caracterizate exhaustiv toate speciile pentru a putea spune că un anumit animal face parte din categoria A sau B. Procedînd intuitiv, pentru ca A să fie constituit ca un set propriu este necesar să se formuleze întrebări asupra oricărui element x din X. În general, dacă setul A este definit prin formularea cu predicate

$A = \{x \mid x \text{ este } \_\_\_\_\_\}$

atunci aceasta incumbă faptul că predicatul nu este precis definit. Cu alte cuvinte se atribuie un conţinut extrasemantic pentru predicat ţinînd cont de stricta interpretare relaţională. Modelul nedeterminat al apartenenţei la clasa introdus de Zadeh, foloseşte noţiunea de subset fuzzy al setului universal X. Pornind de la acesta subsetul A este definit ca $A = \{x \mid u_A(x)\}$ unde funcţia de apartenenţă u_A poate lua valori în intervalul [0, 1], aşa că pentru orice $x \in X$, se obţine

$0 <= u_A(x) <= 1$.

Faţă de situaţia clasică apartenenţa la un subset nu este reprezentată prin valorile precise 0 sau 1. Dacă X reprezintă setul tuturor speciilor, orice element x din X are valoarea $u_B(x)$ indicînd apartenenţa la subsetul B. Din acest punct de vedere x este mai sigur în specia B decît y dacă $u_B(x) > u_B(y)$. Funcţia de apartenenţă determină o caracterizare cantitativă a calităţii.

Exemplu: Se consideră setul X al roboţilor aparţinînd diferitelor corporaţii. Aceşti roboţi formează setul $X = \{a, b, c\}$. Se doreşte caracterizarea setului A al roboţilor inteligenţi. Într-o manieră tranşantă se poate spune că robotul a aparţine clasei A dacă este considerat inteligent sau nu aparţine clasei dacă este complet neinteligent. Utilizarea seturilor fuzzy permite gradarea inteligenţei pe baza funcţiei de apartenenţă. Cel mai inteligent robot este a, cu cea mai mare valoare a funcţiei $u_A(a)$. Intuitiv membrii cu valoare apropiată de 1/2 sînt cei mai dificil de apreciat, adică certitudinea apartenenţei sau neapartenenţei la clasă este cel mai greu de decis. Dacă mulţimea A ce corespunde caracterizării "este inteligent" este:

$A = \{a/0.2, b/0.9, c/0.4\}$

se poate afirma că robotul b este cel mai inteligent.

Dacă $u_A(x) = 1$ atunci se poate scrie ca $x \in A$ respectiv dacă $u_A(x) = 0$ $x \notin A$. Dacă însă $0 < u_A(x) =< 1$, se scrie $x \in A$. Pornind de la teoria de bază se poate spune pentru un set A că este un subset al lui B (adică $A \subset B$) dacă pentru orice x din X este valabilă inegalitatea $u_A(x) <= u_B(x)$.

Despre două seturi fuzzy se spune că sînt egale dacă $u_A(x) = u_B(x)$, şi se notează $A = B$. Pentru operaţiile cu seturile fuzzy între care sînt luate în consideraţie intersecţia, reuniunea, diferenţa este necesară definirea modului de calcul al funcţiei de apartenenţă. Astfel:

$u_{A \cup B} = \max(u_A, u_B)$

$u_{A \cap B} = \min(u_A, u_B)$

$u_{A-B} = \min(u_A, 1 - u_B)$

Faţă de logica clasică în care două valori de adevăr sînt luate în consideraţie, în teoria fuzzy este necesară operarea într-o logică cu mai multe valori. Astfel se cunoaşte logica cu trei valori Kleene, Lukasiewicz, Bochvar, logică ce poate fi extinsă asupra predicatelor fuzzy (Multi-Valued-Logic; MVL). Se defineşte un cadru cu mai multe valori L, ca un set nevid M împreună cu setul relaţiilor fuzzy pe M (unde la n valori fuzzy pe M se obţine o funcţie de la M^n la [0, 1]). Conectivele logice MVL se vor calcula în următorul mod:

(i) $\neg r = 1 - r$

(ii) $r_1 \wedge r_2 = \text{Min}(r_1, r_2)$

(iii) $r_1 \vee r_2 = \text{Max}(r_1, r_2)$

(iv) $r_1 \to r_2 = \text{Min}(1, 1 - r_1 + r_2)$

(v) $\underset{i \in I}{\wedge} r_i = \underset{i \in I}{\text{Min}}(r_i)$

(v) $\underset{i \in I}{\vee} r_i = \underset{i \in I}{\text{Max}}(r_i)$.

Ultimele două introduc specificitatea cuantificatorilor universal şi respectiv existenţial. Cu aceste precizări semantica este similară cu cea din cazul pentru logica cu trei valori. Fie A şi B formule în L la care se poate asocia un număr real în intervalul [0, 1] după cum urmează:

$[A \wedge B] = [A] \wedge [B]$

$[\neg A] = \neg([A])$

$[A \vee B] = [A] \vee [B]$

$[A \to B] = [A] \to [B]$

$[\forall x A] = \underset{m \in M}{\wedge} [A(m)]$

$[\exists x A] = \underset{n \in M}{\vee} [A(m)]$

unde prin $[A]$ s-a notat pentru subsetul fuzzy din intervalul [0, 1], funcţia sa de apartenenţă.

2.5.2. Logica fuzzy

În logica fuzzy (FL) setul valorilor de adevăr în intervalul [0, 1] este înlocuit cu subsetul fuzzy al acelui set. Zadeh, autorul logicii fuzzy nu permite totuşi toate subseturile fuzzy. Valorile de adevăr sînt presupuse a fi numărabile. Un astfel de set are forma

VA = (adevărat, fals, neadevărat, foarte adevărat, nu foarte adevărat, mai mult sau mai puţin adevărat, destul de adevărat,...).

Fiecare element al acestui set reprezintă un subset pe intervalul [0, 1]. Dacă U_{max} este membru al subsetului fuzzy adevărat se pot genera funcţiile de apartenenţă pornind de la acesta. Atunci funcţiile de apartenenţă pentru ceilalţi membri din VA se obţin

$U_{\text{fals}}(v) = U_{\text{adevărat}}(1 - v)$

$U_{\text{neadevărat}}(v) = 1 - U_{\text{adevărat}}(v)$

$U_{\text{foarte adevărat}}(v) = (U_{\text{adevărat}}(v))^2$ etc.

În acest mod înţelesul valorilor lingvistice este crucial dependentă de înţelesul asociat pentru adevărat. Zadeh motivează această alegere prin specificul zonei de interes şi deci în consecinţă înţelesul lingvistic este asociat, introducînd noţiunea de aproximaţie lingvistica. Pentru a ilustra această noţiune se consideră următorul exemplu:

Exemplu
- a este mic
- a şi b sînt aproximativ egale
- b este mai mic sau mai puţin mic

În logica fuzzy "*a este mic*" este interpretat ca asignînd un predicat fuzzy ca o valoare a variabilei ce corespunde atributului corespunzător lui a

înălţime (a) = mic

unde înălţimea este atributul implicat. În termenii unei ecuaţii a doua afirmaţie se scrie

(înălţime (a), înălţime (b)) = aproximativ egal.

Se va obţine utilizînd regulile de mai sus

$b = LA$ [mic @ aproximativ egal]

unde @ este combinarea relaţiilor fuzzy dată prin

U_{mic} @ aproximativ egal (b) = $\text{Max}[U_{\text{mic}}(x) \wedge U_{\text{aproximativ egal}}(x, b)]$

cu x toate obiectele domeniului.

În concluzie, după Zadeh şi Bellman (1976) consecinţa este dată de setul premiselor şi depinde în mod esenţial de sensul ataşat seturilor fuzzy care apar în premise.

Se pot enumera principalele facilităţi ale logicii fuzzy Haack:

(i) valorile de adevăr sînt fuzzy;
(ii) setul valorilor de adevăr lingvistice nu este închis;
(iii) valorile de adevăr sînt subiective şi locale;
(iv) validitatea poate fi caracterizată semantic;
(v) adecvanţa pentru o mare serie de aplicaţii.

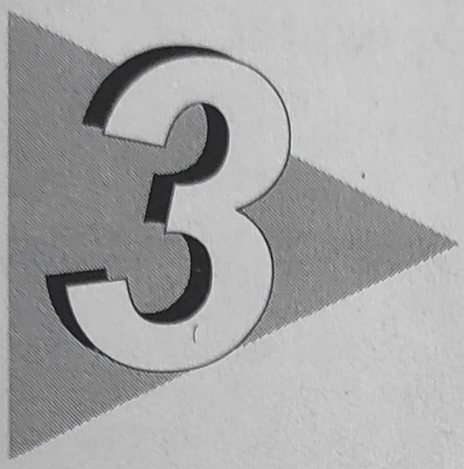
2.5.3. Sisteme expert ce utilizează logica fuzzy

În problemele de raţionare pornind de la date incerte sau incomplete sînt utilizate tehnici similare cu cele fuzzy. Astfel în MYCIN se introduce noţiunea de implicaţie aproximativă utilizînd numere numite factori de certitudine păstrate în baza de cunoştinţe. De exemplu în regula

If the infection is primary-bacterian (and)
the site of the culture is one the sterile sistes (and)
the suspected portal of entry of the organism is the gastrointestinal tract
Then there is suggestive evidence (.7) that the identity of the organism is bacteroides.

Numărul .7 este factorul de certitudine. În acest mod raţionarea este vagă sau inexactă şi este indicată de o valoare numerică. Operatorul pentru conjuncţie în MYCIN este minimizare conform fuzzy, pe cînd disjuncţia este realizată prin interpretare Bayesiana. După modul de utilizare a celor două tehnici o serie de critici pot fi aduse sistemului,

PROSPECTOR utilizează trei tipuri de relaţii în reţeaua inferenţială: relaţii logice, relaţii plauzibile şi relaţii contextuale. Aplicaţii ale teoriei fuzzy sînt găsite în relaţiile logice. O serie de alte aplicaţii în sisteme de diagnosticare cum sînt cele dezvoltate de Saitta şi Torasso, Whalen şi Schott, Soula şi Sanchez fac apel la logica fuzzy.



ARHITECTURI DE SISTEME EXPERT

Aşa cum se deduce din prezentările făcute în capitolul 1 S.E. acoperă o clasă foarte mare de programe ce au o serie de caracteristici comune dar diferă în multe privinţe faţă de programele clasice. Spunînd că există o separaţie netă între cunoştinţe şi mecanismul de raţionare, se poate afirma că nici o cunoştinţă specializată nu participă la fundamentarea mecanismului de inferenţă. De asemenea, baza de cunoştinţe are o serie de caracteristici ce oferă independenţa cunoştinţelor, cum sînt:

- granularitatea cunoştinţelor (independenţa acestora);
- ordinea în care sînt stocate cunoştinţele în baza de cunoştinţe nu influenţează rezultatele;
- modificarea elementelor bazei de cunoştinţe nu are implicaţii catastrofale asupra derulării programului;
- sistemul expert poate fi caracterizat ca o nouă modalitate de programare declarativă, programul este scris ca un ansamblu de specificaţii independente unele de altele, numite şi elemente de cunoaştere ce sînt legate între ele dinamic printr-o procedură de inferenţă.

O caracteristică a sistemelor expert este cea privind competenţa acestora de a furniza explicaţii asupra raţionamentelor întreprinse pentru ajungerea la rezultat, explicaţii ce trebuiesc să fie exprimate într-un limbaj cît mai apropiat de limbajul natural. Întrucît problemele a căror rezolvare este cerută sînt similare cu problemele pe care le rezolvă expertul, şi raţionamentele făcute acestora trebuiesc să corespundă. Multitudinea problemelor determină şi volumul deosebit de mare al bazei de cunoştinţe însă, un S.E. trebuie să fie capabil de a rezolva în aceeaşi măsură şi probleme ce sînt afectate de cunoaştere incertă şi incompletă. În astfel de situaţii se pot utiliza cunoştinţe euristice ce permit găsirea soluţiei potrivite fără ca aceasta să fie nepărat soluţia optimă.

Aşa cum se deduce din prezentările făcute în capitolul 1 S.E. acoperă o clasă foarte mare de programe ce au o serie de caracteristici comune dar diferă în multe privinţe faţă de programele clasice. Spunînd că există o separaţie netă între cunoştinţe şi mecanismul de raţionare, se poate afirma că nici o cunoştinţă specializată nu participă la fundamentarea mecanismului de inferenţă. De asemenea, baza de cunoştinţe are o serie de caracteristici ce oferă independenţa cunoştinţelor, cum sînt:

- granularitatea cunoştinţelor (independenţa acestora);
- ordinea în care sînt stocate cunoştinţele în baza de cunoştinţe nu influenţează rezultatele;
- modificarea elementelor bazei de cunoştinţe nu are implicaţii catastrofale asupra derulării programului;
- sistemul expert poate fi caracterizat ca o nouă modalitate de programare declarativă, programul este scris ca un ansamblu de specificaţii independente unele de altele, numite şi elemente de cunoaştere ce sînt legate între ele dinamic printr-o procedură de inferenţă.

O caracteristică a sistemelor expert este cea privind competenţa acestora de a furniza explicaţii asupra raţionamentelor întreprinse pentru ajungerea la rezultat, explicaţii ce trebuiesc să fie exprimate într-un limbaj cît mai apropiat de limbajul natural. Întrucît problemele a căror rezolvare este cerută sînt similare cu problemele pe care le rezolvă expertul, şi raţionamentele făcute acestora trebuiesc să corespundă. Multitudinea problemelor determină şi volumul deosebit de mare al bazei de cunoştinţe însă, un S.E. trebuie să fie capabil de a rezolva în aceeaşi măsură şi probleme ce sînt afectate de cunoaştere incertă şi incompletă. În astfel de situaţii se pot utiliza cunoştinţe euristice ce permit găsirea soluţiei potrivite fără ca aceasta să fie nepărat soluţia optimă.

3.1. Componentele sistemelor expert

Toate aceste caracteristici determină o structură specifică pentru sistemele expert, structură ce poate fi grupată în jurul a trei module principale, module ce determină şi ceea ce se numeşte *sistem esenţial:*

a) Baza de cunoştinţe este reprezentată ca o structură de date ce conţine ansamblul cunoştinţelor specializate introduse de către expertul uman. Cunoştinţele stocate în baza de cunoştinţe sînt în principal descripţii de obiecte în conjuncţie cu relaţiile dintre acestea. Baza de cunoştinţe face parte din sistemul cognitiv, cunoaşterea fiind memorată într-un spaţiu special organizat. Forma de stocare a informaţiei experte trebuie să asigure căutarea pieselor de cunoaştere specificate direct prin simboluri identificatoare, căutarea pieselor de cunoaştere referite indirect prin proprietăţi asociate sau valori atribuite acestora, căutarea pieselor de cunoaştere prin inferenţe sau lanţuri inferenţiale ce pornesc de la alte piese de cunoaştere sau de la proprietăţi asociate acestora, menţinerea bazei de cunoştinţe în concordanţă cu evoluţia domeniului de expertiză. În capitolul 4 se vor trata principalele modalităţi de reprezentare a cunoştinţelor în baze de cunoştinţe.

b) Mecanismul de inferenţă reprezintă noutatea sistemelor expert. El preia cunoştinţele din baza de cunoştinţe ce sînt utilizate pentru construirea raţionamentului. Mecanismul de inferenţă urmăreşte o serie de obiective majore cum sînt: alege strategia de control funcţie de problema curentă ce o are de rezolvat, elaborează planul de rezolvare a problemei după necesitate, execută comutarea de la o strategie de control la alta, execută acţiunile prevăzute în planul de rezolvare, constituie informaţiile de control pentru mecanismele fundamentale ale mecanismului de inferenţă. Cu toate că în esenţă mecanismul de inferenţă este constituit dintr-un ansamblu de proceduri în sensul obişnuit al termenului, modul în care utilizează cunoştinţele nu este prevăzut prin nici un program, acesta fiind condiţionat de datele şi cunoştinţele pe care le posedă.

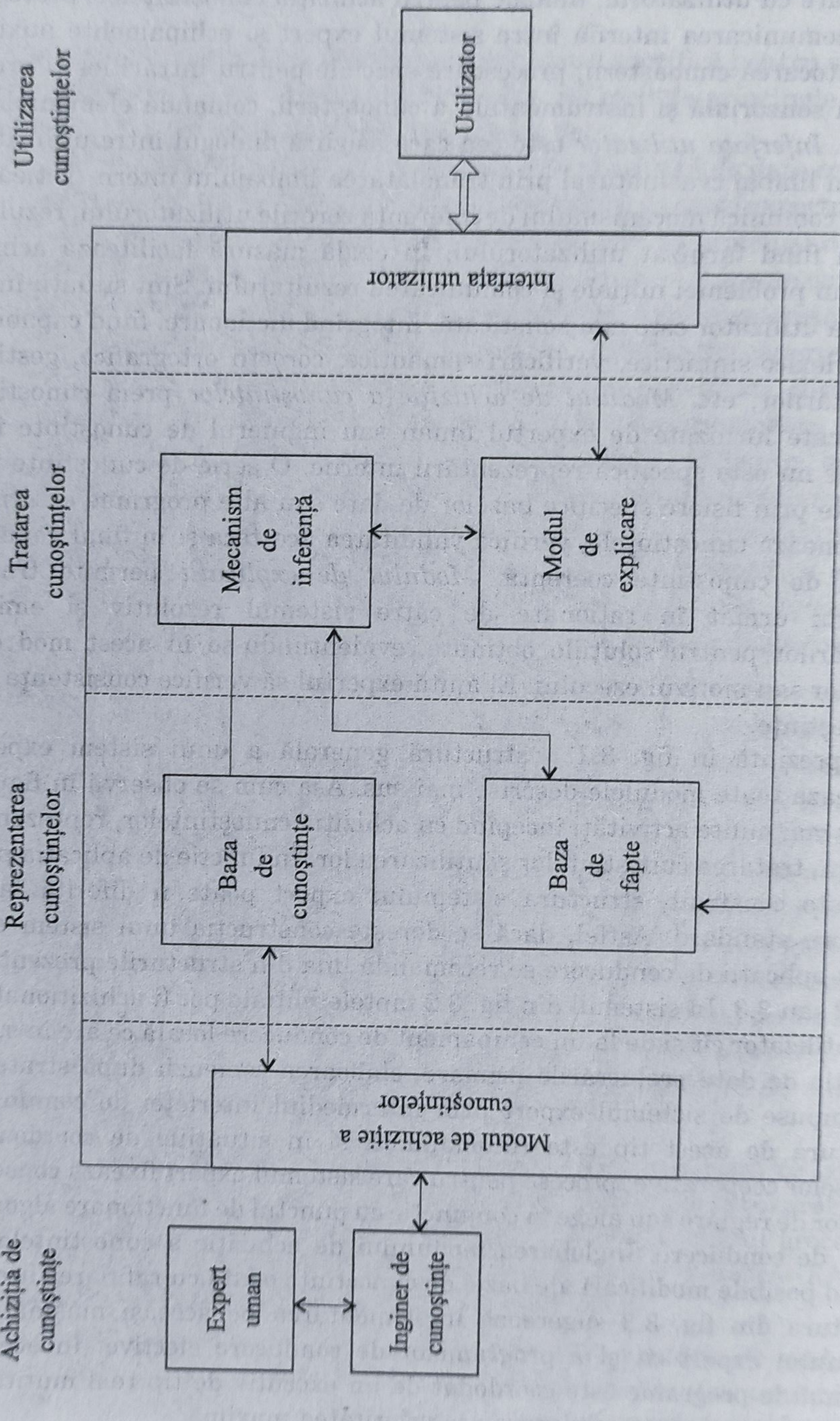
c) Baza de fapte este reprezentată de o memorie auxiliară ce conţine toate datele utilizatorului (faptele iniţiale ce descriu enunţul problemei de rezolvat), şi rezultatele intermediare produse în cursul procedurii de deducţie. Conţinutul bazei este păstrat într-o memorie RAM şi poate fi conservată doar la cererea utilizatorului.

Pe lîngă aceste module un sistem expert mai conţine o serie de module ce asigură comunicarea cu operatorul şi expertul uman. Modulul de comunicaţie este destinat furnizării interfeţelor specifice pentru utilizatorii sistemului

expert cît şi pentru achiziţia de cunoştinţe. În compunerea acestui modul intră procesoare specifice pentru limbajele de comunicare înglobînd limbaje de comunicare cu utilizatorul, limbaje pentru achiziţia cunoştinţelor, procesoare pentru comunicarea internă între sistemul expert şi echipamente auxiliare pentru stocarea cunoaşterii, procesoare speciale pentru intrări ieşiri grafice, achiziţia senzorială şi instrumentală a cunoaşterii, comanda elementelor de execuţie. *Inferfaţa utilizator* este cea care asigură dialogul între utilizator şi sistem în limbaj cvasinatural prin translatarea limbajului intern. Tot ea este cea care comunică mecanismului de inferenţă cererile utilizatorului, rezultatul acestora fiind furnizat utilizatorului. În egală măsură facilitează achiziţia enunţului problemei iniţiale şi comunicarea rezultatului. Sînt situaţii în care interfaţa utilizator este mai sofisticată, integrînd dicţionare, fiind capabilă de analize lexico-sintactice, verificări semantice, corecţii ortografice, gestiunea prescurtărilor, etc. *Modulul de achiziţie a cunoştinţelor* preia cunoştinţele specializate furnizate de expertul uman sau inginerul de cunoştinţe într-o formă ce nu este specifică reprezentării interne. O serie de cunoştinţe pot fi furnizate prin fişiere specifice bazelor de date sau alte programe externe. El recepţionează cunoştinţele, verifică validitatea acestora şi în final generează o bază de cunoştinţe coerentă. *Modulul de explicaţii* permite trasarea drumului urmat în raţionare de către sistemul rezolutiv şi emiterea justificărilor pentru soluţiile obţinute, evidenţiindu-se în acest mod cauza greşelilor sau motivul eşecului. El ajută expertul să verifice consistenţa bazei de cunoştinţe.

Se prezintă în fig. 3.1 o structură generală a unui sistem expert ce înglobează toate modulele descrise mai sus. Aşa cum se observă în figură se disting mai multe activităţi începînd cu achiziţia cunoştinţelor, reprezentarea acestora, tratarea cunoştinţelor şi utilizarea lor. În funcţie de aplicaţia pentru care este construit, structura sistemului expert poate fi diferită faţă de structura standard. Astfel, dacă se doreşte construcţia unui sistem expert pentru aplicaţii de conducere se recomandă una din structurile prezentate în fig. 3.2 sau 3.3. În sistemul din fig. 3.2 faptele iniţiale pot fi achiziţionate atît de la utilizator cît şi de la un echipament de conducere locală ce are în sarcină achiziţia de date prelucrările primare, elaboarea comenzii după strategii ce sînt impuse de sistemul expert prin intermediul interfeţei de comunicaţie. Structura de acest tip este recomandată şi în situaţiile de coordonare a proceselor cooperative, procese pentru care sistemul expert fixează consemnul buclelor de reglare sau alege în conjuncţie cu punctul de funcţionare algoritmul optim de conducere. Înglobarea modulului de achiziţie a cunoştinţelor face oricînd posibile modificări ale bazei de cunoştinţe odată cu rafinarea acestora. Structura din fig. 3.3 sugerează implementarea pe aceeaşi maşină atît a sistemului expert cît şi a programelor de conducere efective. În acest caz sistemul de programe este coordonat de un executiv de tip real mutitasking pentru care taskul de conducere are prioritatea maximă.

▼ ***Fig. 3.1. Structura generală a unui sistem expert***



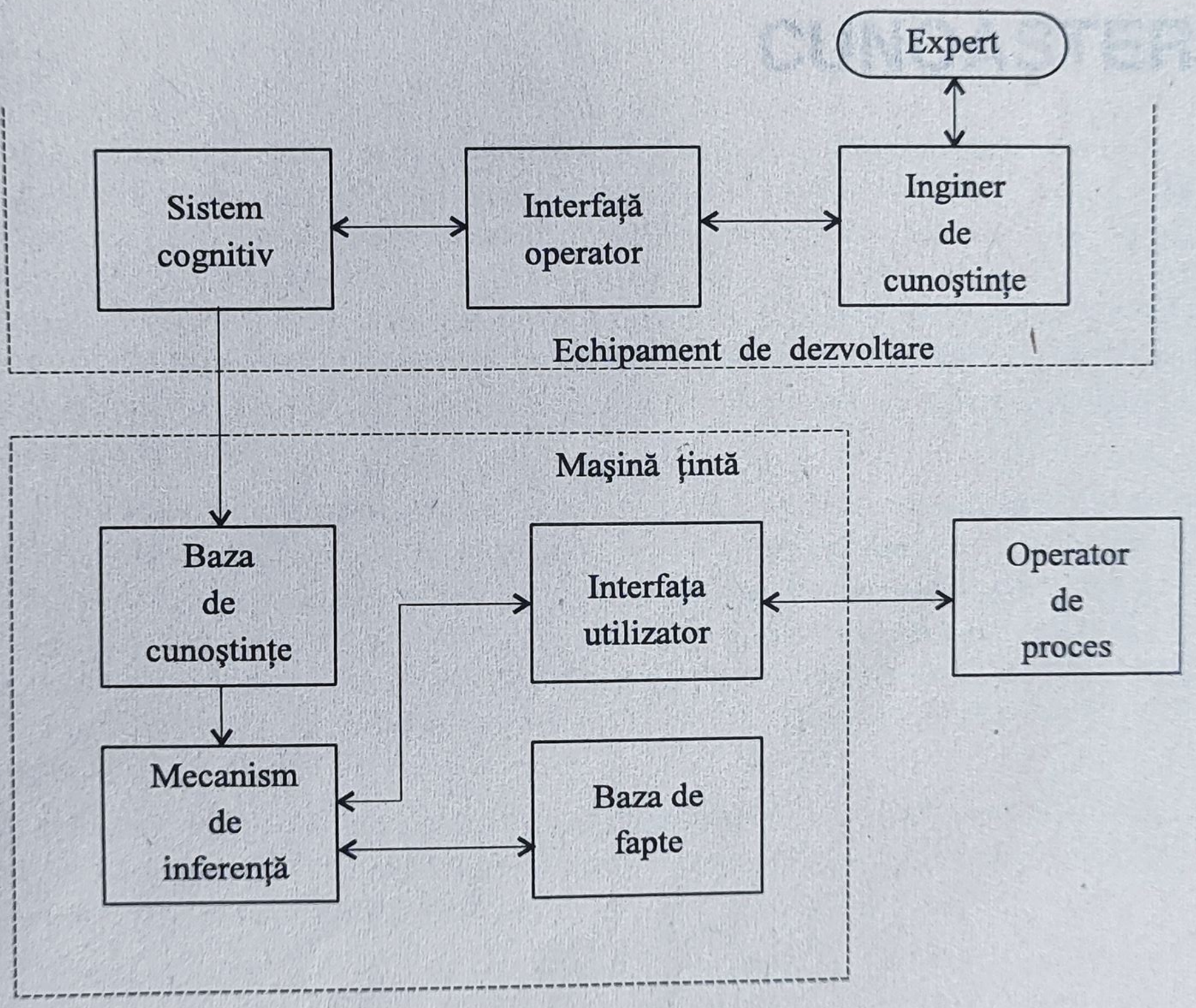
▼ ***Fig. 3.2. Structura de sistem expert în comunicaţie cu echipament de conducere locală***

▼ ***Fig. 3.3. Structura de sistem expert cuplat cu algoritmi de conducere***

3.2. Sisteme Expert pentru aplicaţii în timp real

Este interesant de observat faptul că implementarea sistemului cognitiv pe un echipament ce lucrează în timp real la nivelul buclelor de reglare (controller) creează probleme din punctul de vedere al memoriei ocupate cît şi al sofware-lui asociat în condiţiile în care principala activitate la acest nivel este cea de conducere. Pe de altă parte sarcinile unui sistem expert la acest nivel sînt mult limitate, datorită universului de cunoaştere la nivelul buclei. În schimb, la acest nivel baza de cunoştinţe se modifică doar la intervale mari de timp (săptămîni, luni).

▼ ***Fig. 3.4. Structura unui sistem expert cu baza de cunoştinţe realizată pe sistem de dezvoltare***



Ca urmare este un bun motiv de a propune o structură care, chiar cu inconvenientul îngheţării pentru o perioadă mai mare de timp a bazei de cunoştinţe oferă o mai mare supleţe sistemului (fig. 3.4). În această situaţie activitatea de achiziţie a cunoştinţelor se execută pe echipamentul de dezvoltare, baza de cunoştinţe rezultată fiind transferată pe maşina ţintă.

4

REPREZENTAREA CUNOAȘTERII

Reprezentarea cunoştinţelor într-un calculator constă în găsirea unei corespondenţe între lumea exterioară şi sistemul simbolic ce permite execuţia raţionamentelor. În acest scop se extrag din observaţiile făcute asupra obiectelor, a faptelor şi fenomenelor, acele caracteristici cărora li se pot asocia semnificaţii determinate în imaginea mentală formată despre lume. Descrierea are ca scop deosebirea imaginii obiectului dat de imaginile celorlalte obiecte ale lumii înconjurătoare la fel cum obiectul fizic se deosebeşte de restul obiectelor lumii reale. Aceasta arată că descrierea reflectă însăşi cunoaşterea despre acel obiect.

Filozofic cunoaşterea poate fi definită ca o reflectare activă în conştiinţă a lumii reale, a esenţialului şi generalului din fenomene. Pot fi reliefate două componente ale cunoaşterii ştiinţifice şi anume: *cunoaşterea reflexivă* bazată pe reflectarea exterioară a faptului ştiinţific şi *cunoaşterea generativă* bazată pe crearea de noi obiecte abstracte, prin constituirea de momente iniţiale pentru procesele de construire a unor noi fapte ştiinţifice şi programe de acţiune.

Cunoaşterea este raportată la un "subiect cunoscător", adică un agent ce are capacitatea de interpretare a unei informaţii pe care o are la dispoziţie prin extragere din memoria proprie.

Cunoaşterea este *empirică* dacă informaţiile despre obiecte, fapte, fenomene, procese, necunoscute este sesizată de subiectul cunoscător prin organele sale senzoriale sau prin intermediul unor aparate sau instrumente.

Cunoaşterea este teoretică dacă se desfăşoară după raţionamente şi judecăţi reliefînd legăturile interne, cauzalitatea, legităţile după care se dezvoltă structurile şi se desfăşoară procesele. Ea se dezvoltă din cunoaşterea empirică, prin analiză, sinteză, deducţie, inducţie, generalizare, particularizare.

Datorită caracterului de reflectare în conştiinţă a realităţii, cunoaşterea are un caracter individual, subiecţii cunoscători pot avea nivele de cunoaştere diferită funcţie de lucrurile reprezentate. Termenul de cunoaştere arată că de fapt reprezintă o cunoaştere *parţială* dacă permite deosebirea părţii reprezentate de restul lumii, respectiv cunoaştere *incompletă* dacă apar situaţii în care părţi dinstincte ale lumii pot avea aceeaşi reprezentare.

În transpunerea de la subiecţii umani la subiecţii cunoscători de tip "program pentru calculator" deci de la psihologia gîndirii la inteligenţa artificială conceptul de cunoaştere îşi conservă calităţile de mai sus. În vederea utilizării unui program cunoaşterea este memorată sub forma unor piese de cunoaştere ce descriu fapte, fenomene, procese, evenimente dintr-o parte a lumii reale ce constituie domeniul de competenţă al programului inteligent. Piesele de cunoaştere alcătuiesc un model al lumii la care programul are acces prin intermediul procedurilor de organizare, clasificare, căutare şi recunoaştere.

Se poate, în final, defini un sistem cognitiv ca totalitatea pieselor de cunoaştere, a modului de stocare şi a procedurilor de acces la acestea.

Problema fundamentală a inteligenţei artificiale este cea de definire a unor metode pentru reprezentarea unei mari cantităţi de cunoştinţe într-o formă ce permite stocarea şi utilizarea eficientă a acestora.

4.1. Noţiuni introductive despre reprezentarea cunoaşterii

Reprezentarea cunoaşterii poate fi conform celor de mai sus considerată ca o relaţie de definiţie $A \Leftrightarrow B$ ce stabileşte o legătură între un simbol identificator A (numele entităţii reprezentate) şi o expresie B formulată în limbajul de reprezentare al sistemului, ce descrie caracteristicile entităţii şi structura sa cu ajutorul unor termeni primari, termeni purtători de semnificaţie ai limbajului. Expresia B este o reprezentare a entităţii A sau a piesei de cunoaştere A în sistemul cognitiv considerat ca gazdă a reprezentării cunoaşterii.

Orice activitate de reprezentare a cunoaşterii are loc după stabilirea ordinii conceptuale pe baza căreia se construieşte metoda efectivă de reprezentare, ce are în vedere stabilirea problemelor relative la piesele de cunoaştere ca obiecte primitive ale sistemului formal, probleme ce pot fi sintetizate ca mai jos:

- stabilirea *sistemului de meta-reprezentare* ce are ca obiectiv asigurarea unor modalităţi adecvate de reprezentare a cunoaşterii despre modelul lumii reale cît şi pentru componentele proprii ale sistemului inteligent;
- stabilirea *sistemului de clasificare* ce urmăreşte gruparea în clase, precum şi ordonarea acestora după criterii ce rezultă din analiza

relevanţei proprietăţilor ce sînt folosite pentru caracterizarea obiectelor ce formează piesele de cunoaştere;

- stabilirea *sistemul de organizare* privitor la gruparea şi ordonarea elementelor şi claselor, avînd în vedere asigurarea realizării unor procese fundamentale cum sînt: acces, căutare, corespondenţă, substituire, inferenţă deductivă, inferenţă inductivă, grupare şi regrupare, organizare şi reorganizare, protecţie.

Se poate apoi trece la enunţuri văzute ca obiecte ale limbajului formal. Sînt necesare reguli privind formarea enunţurilor din piese de cunoaştere, stabilirea sistemului teoretic prin definirea unui set de enunţuri-axiomă şi a regulilor de inferenţă. Se prezintă mai jos principalele probleme relative la piesele de cunoaştere.

4.1.1. Sistemul de meta-reprezentare

Prin meta-reprezentare se înţelege acea reprezentare a cunoaşterii despre cunoaşterea însăşi şi despre reprezentările ei. Ca urmare domeniul de competenţă al meta-reprezentării este chiar *cunoaşterea abstractă*. Principalele piese de meta-cunoaştere sînt:

- *meta-cunoaşterea reprezentării obiectelor* ce cuprinde descrierea schemelor de reprezentare utilizate pentru piesele de cunoaştere de tip obiectual;
- *meta-cunoaşterea reprezentării funcţiilor* ce cuprinde descrierea şabloanelor funcţionale destinate pieselor de cunoaştere de tip procedural;
- *meta-cunoaşterea reprezentării relaţiilor* ce cuprinde descrierea schemelor de reprezentare a pieselor de cunoaştere, reprezentînd relaţii între piese de cunoaştere de tip obiectual;
- *meta-cunoaşterea reprezentării regulilor de inferenţă* ce cuprinde descrierea schemelor de reprezentare a regulilor;
- *meta-cunoaşterea reprezentării strategiilor de raţionamente* ce cuprinde descrierea meta-regulilor şi a formalismelor de reprezentare a planurilor;
- *meta-cunoaşterea auto-reprezentării* ce cuprinde descrierea sistemului inteligent în el însuşi.

Se disting situaţii specifice funcţie de modul în care se dispune de informaţia necesară pentru reprezentarea expresiei definitorii a piesei de cunoaştere, fapt ce determină meta-cunoaşterea corespunzătoare:

- *reprezentarea descriptivă* realizată prin caracterizarea directă a componentelor, structurii şi proprietăţilor piesei de cunoaştere;

- *reprezentarea comparativă* în care pentru a se reprezenta piesa de cunoaştere A expresia B conţine o referinţă la o piesă auxiliară numită şi prototip şi o informaţie ce marchează diferenţele între aceasta şi piesa prototip. Astfel pentru reprezentarea piesei de cunoaştere A se reprezintă o referinţă B la o unitate auxiliară C şi o informaţie suplimentară D, care punctează diferenţele între A şi C;
- *reprezentarea constructivă* (generativă) în care expresia definitorie cuprinde o relaţie generatoare a elementelor componente, structurii şi proprietăţii piesei de cunoaştere;
- *reprezentarea enumerativă* în care expresia definitorie cuprinde referiri la elementele unei mulţimi care este definită ca piesă de cunoaştere.

Ca strategie generală se poate afirma că în general se optează pentru una din reprezentări, dar o reprezentare completă este oferită doar printr-o combinaţie finită a metodelor de reprezentare, deoarece cunoaşterea despre un obiect este dată parţial în termeni descriptivi, parţial în termeni comparativi faţă de un obiect cunoscut, parţial în termeni generativi şi enumerativi. Este de dorit ca sistemul să accepte completarea reprezentării folosind mecanismele inferenţiale asociate sistemului cognitiv.

4.1.2. Sistemul de clasificare

Ordinea conceptuală se realizează cu ajutorul unui sistem de clasificare ce urmăreşte stabilirea unui număr finit de clasificări a pieselor de cunoaştere care să constituie ipostaze diferite ale sistemului de interpretare. Clasificarea, presupune stabilirea unor grupări numite clase, pe baza unei similarităţi denumită şi criteriu de clasificare, proces ce se efectuează în două etape:

- *clasificare conceptuală* prin care proprietăţile obiectului sînt analizate din punct de vedere al relevanţei în raport cu obiectele utilitare ale sistemului, pentru a fi grupate apoi în criterii;
- *clasificare instanţială* prin care piesele de cunoaştere sînt plasate în clase potrivit criteriului de clasificare stabilit pentru fiecare piesă de cunoaştere.

În general, prin criteriu de clasificare se înţelege stabilirea categoriilor logice, a proprietăţilor ce determină pe mulţimea pieselor de cunoaştere o partajare în clase. Stabilirea criteriilor de clasificare reprezintă o fază conceptuală de mare importanţă în reprezentarea cunoaşterii. Clasificarea este orientată către obiectivul utilitar al aplicaţiei. Astfel pentru un sistem expert independent de domeniul de aplicaţie limbajul de reprezentare a cunoaşterii trebuie să permită comunicarea criteriilor de clasificare ca piese de cunoaştere privind sistematizarea bazei de cunoştinţe în funcţie de obiectivele generale ale sistemului. O clasă este constituită prin predicaţie,

elementele sale determină că predicatul ce semnifică proprietăţile din criteriul de clasificare să ia valoarea adevărat.

Se spune despre o clasă că este o *extensiune* a unui predicat a cărui instanţiere este proprietatea luată drept criteriu de clasificare. Un criteriu de clasificare care nu poate decide satisfacerea condiţiilor de unicitate a clasificării este un criteriu de grupare sau *clustering*. Ca urmare un element poate să aparţină mai multor clase. Ca exemplu, piesa de cunoaştere reprezentînd automobilul DACIA aparţine unei singure clase şi anume clasa autovehiculelor. În acelaşi timp aparţine mai multor grupări cum sînt: grupul autoturismelor, grupul autovehiculelor cu motor Otto, grupul autovehiculelor de fabricaţie românească, etc. Prin apartenenţa sa la clasa autovehiculelor DACIA moşteneşte acele proprietăţi ce au servit drept criteriu al clasificării sale ca automobil: destinaţie - transport, consumă - combustibil, are sistem de rulare, etc. Ataşarea la grupări nu modifică proprietăţile ce îl definesc ca membru al clasei ci asigură o serie de noi trăsături de importanţă secundară a căror influenţă asupra procesului de interpretare sînt specificate prin procedeele de ataşare procedurală.

4.1.3. Sistemul de organizare

Prin acest sistem se înţelege activitatea de ordonare şi grupare a pieselor de cunoaştere după criterii funcţionale ce rezultă din analiza modalităţilor de preluare din baza de cunoştinţe a unor piese de cunoaştere necesare unor procese în sistemul rezolutiv. În general această activitate nu dispune de abordări teoretice care să permită fundamentarea de soluţii ce se vor utiliza în aplicaţii practice. În diferite programe aplicative s-au utilizat soluţii proprii dependente de domeniul de competenţă al aplicaţiei. Se pot enumera cîteva dintre principalele criterii conceptuale de organizare:

- accesul la piesele de cunoaştere şi la componentele lor;
- modul de execuţie a proceselor inferenţiale;
- existenţa relaţiilor de ordine;
- protecţia împotriva manevrelor neautorizate;
- asigurarea consistenţei bazei de cunoştinţe.

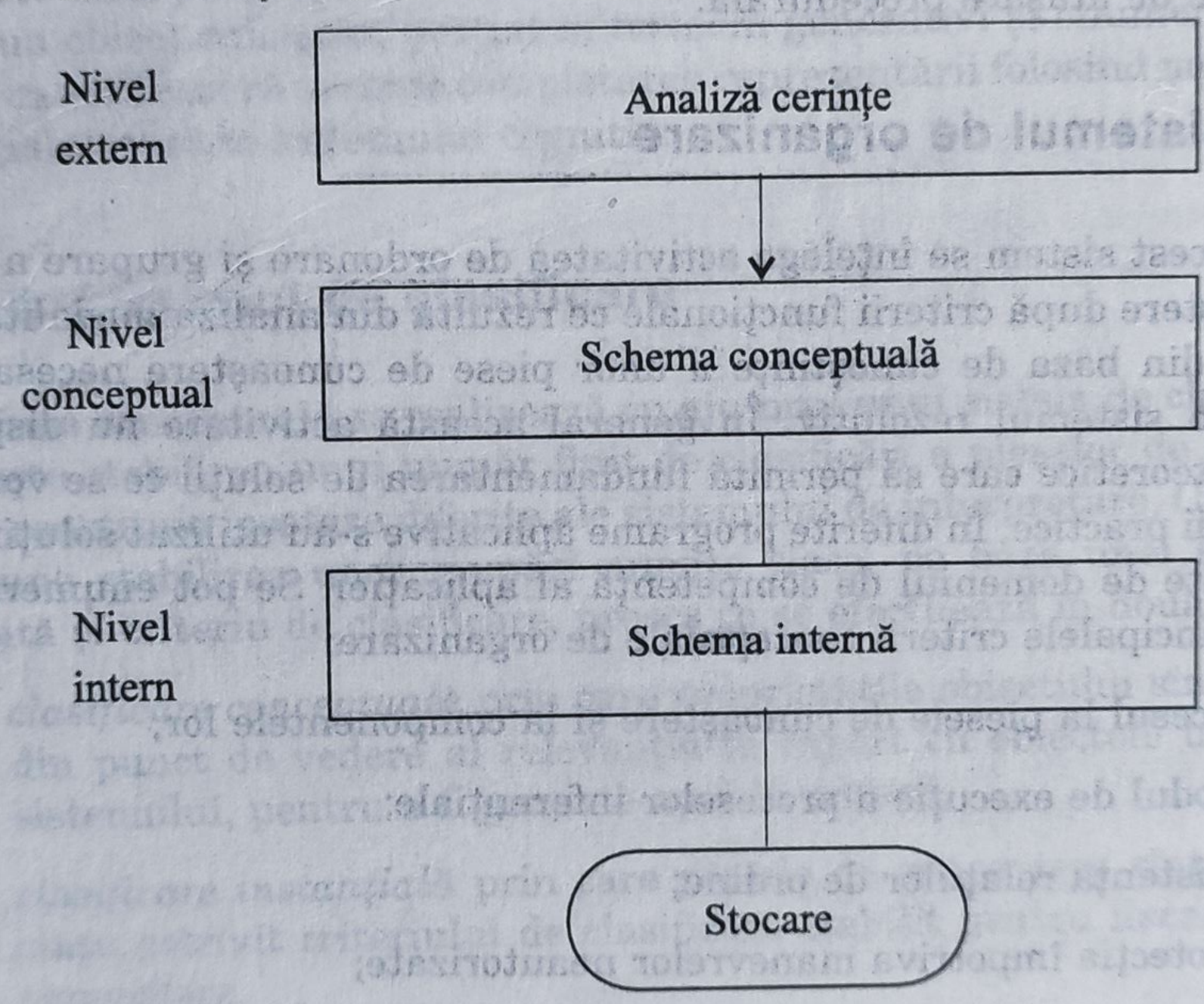
Faţă de programele scrise în limbajele de programare ce utilizează metode de organizare şi acces standard, limbajale destinate inteligenţei artificiale oferă extrem de puţine facilităţi de organizare a bazelor de cunoştinţe, lăsînd în general această sarcină în seama realizatorilor de aplicaţii. Este poate unul din motivele pentru care aplicaţiile de inteligenţă artificială nu au cunoscut pînă în prezent o dezvoltare asemănătoare aplicaţiilor în tehnicile tradiţionale. Se poate remarca faptul că procedeele specifice structurilor de date în programarea clasică nu sînt disponibile aici.

4.2. Metode de reprezentarea cunoaşterii

Reprezentarea cunoaşterii urmăreşte descrierea universului în care sistemul efectuează raţionamente sub formă de entităţi corespunzătoare indivizilor şi sub formă de simboluri pentru relaţiile între aceştia. Deoarece în timp cunoaşterea despre indivizi şi relaţiile dintre aceştia poate să se modifice nu este posibilă o reprezentare declarativă a stării iniţiale şi se pune problema înregistrării enumerative a succesiunilor de stare cît şi a mecanismului ce asigură transformarea stării iniţiale spre stările succesive.

O arhitectură de reprezentare a cunoştinţelor poate fi considerată ca fiind constituită pe trei nivele. Cele trei nivele sînt stabilite de locul în care se judecă cunoştinţele. O schemă de reprezentare a cunoştinţelor pe trei nivele este reprezentată în fig. 4.1.

▼ ***Fig. 4.1. Arhitectura pe trei nivele de reprezentarea datelor***



Nivelul intern este constituit din schema internă ce descrie structura de stocare fizică a cunoştinţelor în baza de cunoştinţe. La acest nivel se descriu detaliile complete ale stocării, precum şi modul de acces la cunoştinţe.

Nivelul conceptual sau schema conceptuală descrie structura întregii baze de cunoştinţe pentru o comunitate de utilizatori. La nivelul conceptual se face o descriere completă a bazei de cunoştinţe, ascunzînd detaliile legate de stocarea fizică, concetrîndu-se asupra descrierii entităţilor, tipurilor de date, relaţiilor dintre ele, precum şi a restricţiilor asociate. Poate fi utilizat cu bune rezultate la acest nivel un model de nivel înalt, sau un model specific de implementare.

Nivelul extern sau nivelul vizual (utilizator), include o colecţie de scheme externe ce descrie baza de cunoştinţe prin prisma diferiţilor utilizatori. Fiecare grup utilizator descrie baza de cunoştinţe prin prisma propriilor interese. Există tendinţa la acest nivel ca grupuri de utilizatori să ascundă detalii de care nu sînt interesate. Şi la acest nivel se pot folosi modele de implementare sau modele de nivel înalt.

Desigur că, în multe sisteme nu există o distincţie netă între cele trei nivele. Cu toate acestea se poate remarca în majoritatea sistemelor cognitive un nivel conceptual puternic ce suplineşte aparent de cele mai multe ori celelalte nivele. De asemenea, se remarcă o contopire în aplicaţii între nivelele conceptual şi extern. În plus arhitectura pe trei nivele reprezintă numai o descriere a datelor ce vor forma baza de cunoştinţe. Grupurile de utilizatori se referă numai la schema externă, deci, sistemul cognitiv este cel ce va trebui să transforme schema externă în schema conceptuală.

Unul de conceptele de bază de reprezentare a cunoştinţelor este cel de entitate. O entitate este un obiect al lumii reale, cu o existenţă independentă. O entitate este un obiect cu existenţă fizică, persoană particulară, automobil, companie, activitate, curs universitar, etc. Orice entitate are o serie de proprietăţi numite atribute ce particularizează entitatea respectivă. De exemplu, pentru o entitate automobil se pot enumera o serie de atribute cum sînt marca, combustibilul utilizat, capacitate cilindrică, etc. Valorile acestor atribute au ca scop identificarea entităţii.

Unele atribute pot fi împărţite în părţi mai mici cu semnificaţie independentă. Un astfel de atribut este un atribut *complex*. Un exemplu este cel al atributului adresa, ce poate fi subdivizat în atributele componente Oraş, Judeţ, Cod poştal, Stradă. La rîndul său atributul Stradă este definit prin atributele Nume stradă, Număr, Scară, Număr apartament. Un alt exemplu este cel al atributului Nume pentru o persoană fizică ce poate fi subdivizat în atributele simple Nume, Iniţială, Prenume. Atributele ce nu sînt compuse se numesc atribute *atomice*. Valoarea atributelor compuse se formează prin compunerea valorilor atributelor atomice.

Multe atribute au valoare unică pentru o entitate particulară şi sînt numite atribute cu o *singură valoare*. Spre exemplu vîrsta unei persoane. Există atribute ce pot lua mai multe valori ca de exemplu gradele didactice într-o universitate, culoarea unui automobil, etc. Acestea sînt atribute cu *mai multe valori*.

Atributele derivate sînt atributele ce se pot determina din relaţiile ce există între ele ca de exemplu, vîrsta unei persoane ce se poate determina din data curentă şi data de naştere. În anumite situaţii o entitate particulară poate să nu aibă valori aplicabile pentru toate atributele, ca de exemplu în cadrul adresei numărul apartamentului cînd este vorba de o casă particulară.

Descrierea atributelor entităţii este numită *schema entităţii* şi specifică o structură comună fixată a entităţilor de acelaşi tip. Setul instanţelor entităţilor individuale la un anumit moment este numită extensie a entităţii.

Fiecare atribut al unei entităţi tip are asociat un set de valori, numit şi *domeniu*, ce specifică valorile posibile pe care le poate lua. De exemplu, vîrsta unui angajat este curpinsă între 16 şi 70 ani. Matematic, atribtul A al entităţii tip E poate fi definit ca o funcţie de la E la setul valorilor posibile ale lui V, sau la toate subseturile lui V.

$$A: E \rightarrow P(V)$$

Valoarea atributului A pentru entitatea e, se va nota ca $A(e)$. Pentru un atribut simplu $A(e)$ este o valoare cu un singur element, un atribut null nu are valoare sau altfel spus are valoarea null. Pentru un atribut compus A setul de valori este format ca produsul cartezian din $P(V_1)$, $P(P_2)$,..., $P(V_n)$ unde V_1, V_2,..., V_n reprezintă setul valorilor unui simplu component al lui A. Deci

$$V = P(V_1) \times P(V_2) \times \ldots \times P(V_n)$$

Între entităţi se pot stabili o serie de relaţii ce pot avea la rîndul lor atribute ce le caracterizează. Dacă valoarea unui atribut este determinată prin combinarea entităţilor participante într-o relaţie instantă, şi nu prin entităţi singure, atunci atributul se va specifica ca un *atribut relaţie.*

Se pot distinge pentru aplicaţiile de inteligenţă artificială următoarele clase de metode de reprezentare:

- *metode logice* - sînt acele metode ce privesc cunoaşterea ca o serie de aserţiuni (enunţuri adevărate) privind cunoştinţele şi relaţiile între ele. Metoda permite folosirea regulilor de inferenţă direct asupra pieselor de cunoaştere din baza de cunoştinţe. Are însă dezavantaje prin soluţiile nesatisfăcătoare de sistematizare a bazei de cunoştinţe, prin inadecvanţa reprezentării cunoaşterii despre acţiuni, precum şi a reprezentării regulilor euristice;
- *metode relaţionale* - sînt metodele prin care cunoaşterea este reprezentată pornind de la relaţiile între obiecte sub formă de grafuri şi reţele. Metodele din această categorie permit organizarea cunoştinţelor funcţie de omogenitatea acestora ce conduc la clase şi sorturi. Metodele relaţionale reprezintă de regulă legăturile dintre obiectele universului ca pe nişte elemente ale cunoaşterii, spre deosebire de metodele logice în care acestea de regulă se deduc;
- *metode procedurale* - în care cunoaşterea este reprezentată sub forma de proceduri ce permit obţinerea stărilor la momentele specificate pornind de la stările iniţiale sau intermediare.

Descrierea formală a unui concept în termenii limbajului de reprezentare, prin care se diferenţiază de alte concepte sau prin care se poate aprecia echivalenţa cu alte concepte similare poartă numele de *definiţie*. Definiţia are ca obiectivitate modelarea obiectelor şi a relaţiilor dintre ele astfel încît să fie specificate caracteristicile esenţiale ale acestora cît şi posibilitatea de a facilita operaţiile în care obiectele şi relaţiile sînt frecvent implicate. Este normal ca pentru un concept să poată fi date mai multe definiţii, fiecare definiţie fiind o descriere parţială dependentă de perspectiva din care este privită.

Altă informaţie privind un concept este dată de relaţiile acestuia cu entităţi de tip *acţiune*. În general un obiect poate să ocupe în cadrul unei acţiuni poziţia de *argument*, deci asupra căruia se acţionează cît şi poziţia de *rezultat* al acţiunii. Se numesc *cazuri* acele participări ale obiectelor la acţiuni ce se deosebesc prin semnificaţie. Pentru un concept dat este relevant să se specifice o listă a acţiunilor în care conceptul reprezentat se constituie ca argument şi o listă a acţiunilor în care acesta se constituie ca rezultat. Pe lîngă posibilitatea efectuării de raţionamente se pot genera obiecte instanţă ale conceptului prin efectuarea operaţiilor ce apar în lista ataşată poziţiei de rezultat pentru acel concept.

Se va numi *generalizare* a unui concept C o definiţie mai puţin restrictivă a acestuia D, în care orice instanţă a conceptului C este şi instanţă a conceptului D, concept mai general. Generalizarea face ca din definiţia conceptului C să se omită unele proprietăţi mai puţin relevante. O serie de proprietăţi sînt exprimate printr-un predicat $P(x, C, D)$ în care x este o instantă de concept, care dacă satisface şi predicatul pentru generalizare va satisface şi definiţia conceptului C.

$$DEF(x, D) \wedge P(x, D, C) \rightarrow DEF(x, C)$$

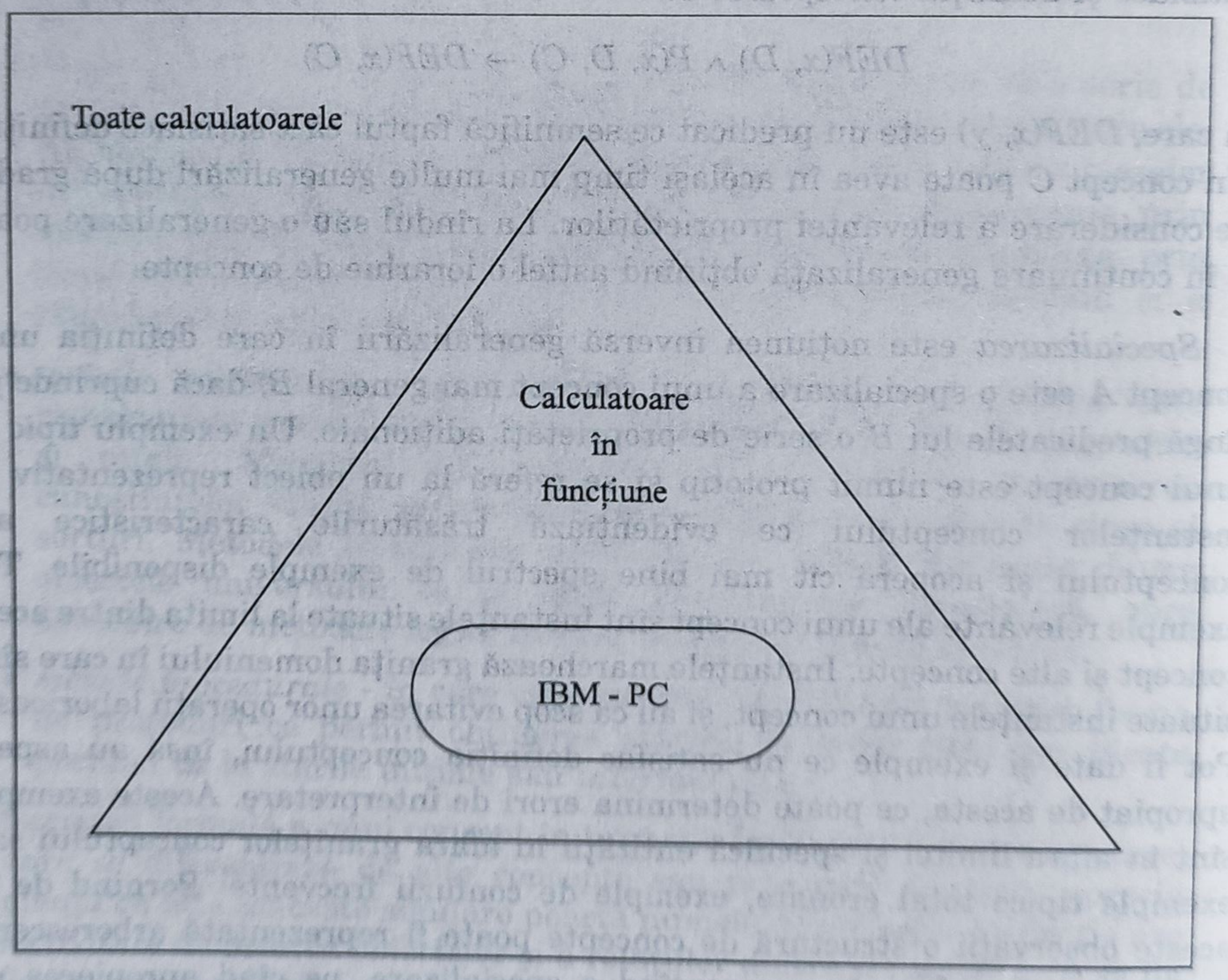
în care, $DEF(x, y)$ este un predicat ce semnifică faptul că x satisface definiţia. Un concept C poate avea în acelaşi timp mai multe generalizări după gradul de considerare a relevanţei proprietăţilor. La rîndul său o generalizare poate fi în continuare generalizată obţinînd astfel o ierarhie de concepte.

Specializarea este noţiunea inversă generalizării în care definiţia unui concept A este o specializare a unui concept mai general B, dacă cuprinde pe lîngă predicatele lui B o serie de proprietăţi adiţionale. Un exemplu tipic al unui concept este numit prototip şi se referă la un obiect reprezentativ al instanţelor conceptului ce evidenţiază trăsăturile caracteristice ale conceptului şi acoperă cît mai bine spectrul de exemple disponibile. Tot exemple relevante ale unui concept sînt instanţele situate la limita dintre acest concept şi alte concepte. Instanţele marchează graniţa domeniului în care sînt situate instanţele unui concept, şi au ca scop evitarea unor operaţii laborioase. Pot fi date şi exemple ce nu satisfac definiţia conceptului, însă au aspect apropiat de acesta, ce poate determina erori de interpretare. Aceste exemple sînt în afara limitei şi specifică entităţii în afara graniţelor conceptului sau exemple tipice total eronate, exemple de confuzii frecvente. Pornind de la aceste observaţii o structură de concepte poate fi reprezentată arborescent, înaintarea spre frunză reprezentînd o specializare, pe cînd apropierea de rădăcină o generalizare a conceptului.

4.2.1. Reprezentarea cunoaşterii în limbajul calculului cu predicate de ordinul întîi

În acest mod de reprezentare piesele de cunoaştere sînt descrie prin expresii ale căror componente sînt formule ale limbajului. Metoda de reprezentare are avantajul introducerii pieselor de cunoaştere direct în sistemul rezolutiv dacă acesta este conceput pe baza unor mecanisme definite în calculul cu predicate de ordinul întîi. Baza reprezentării este dată de caracteristica de funcţie propoziţională a predicatului. O piesă de cunoaştere exprimată în limbaj natural este descompusă în propoziţii elementare adecvate numite *"aserţiuni"* ce specifică fapte (proprietăţi, relaţii legate de piesa de cunoaştere). O propoziţie elementară este generată de un predicat cu un număr finit de locuri în care se specifică variabile formale sau obiecte în mulţimea suport.

▼ ***Fig. 4.2. Plasarea obiectelor în univers***



Se poate spune că acest mod de reprezentare este constituit din două etape şi anume:

- *reprezentarea propoziţională* - piesele de cunoaştere sînt descompuse în aserţiuni legate prin conectivele logice specifice calculului propoziţional;
- *reprezentarea predicativă* - în care fiecare aserţiune este descompusă în componentele sale, predicate sau obiecte.

Calculul cu predicate reprezintă un model al raţionării cu două valori, model ce implică analiza conexiunilor între clauze. Modul de raţionare este asemănător raţionării umane. Fie următoarea inferenţă simplă:

Toate calculatoarele sînt în funcţiune; (i)
IMB PC este un calculator; (ii)
Deci, IMB PC este în funcţiune (iii)

Inferenţele în calculul propoziţional rezultă din structura formulării. O propoziţie atomică este cea mai scurtă propoziţie cu înţeles în calculul propoziţional. Oricum, inferenţa din formularea (i) necesită analiza propoziţiilor primare. În special relaţia subiect predicat trebuie luată în consideraţie. Calculul cu predicate este o extensie a calculului propoziţional şi constă în faptul că acesta extinde forma propoziţională.

Pentru descrierea calcului cu predicate sînt necesare noi definiţii şi notaţii. Un *termen* este o variabilă sau o constantă, dată prin nume. Variabila x, unde x semnifică un termen este un termen. În (ii) constanta IMB PC reprezintă o instanţă a lui x, adică un tip particular de calculator. Ori de cîte ori o variabilă este egală cu o constantă se spune că variabila este instanţiată. Notaţia *CALCULATOR*(y) indică faptul că y este un calculator, iar notaţia *FUNCŢIONEAZĂ*(x) indică proprietatea lui x de a fi în funcţiune. Ambele *CALCULATOR*(y) şi *FUNCŢIONEAZĂ*(x) sînt numite predicate şi au aritatea 1, întrucît fiecare indică un singur argument. În general predicatele sînt relaţii cu aritate n, indicînd prin aceasta faptul că ele conţin n argumente. Atunci cînd toate variabilele sau argumentele predicatului sînt instanţiate, expresia rezultantă este o propoziţie a calculului propoziţional. Propoziţia poate fi adevărată sau falsă. În situaţia în care $n = 0$ se spune că predicatul însuşi este o propoziţie.

Gramatical, predicatul instanţiat *CALCULATOR* (IBM PC) reprezintă declaraţia "IMB PC este un calculator" care este fie adevărată, fie falsă. În exemplul dat, prima afirmaţie (i) este o propoziţie declarativă ce conţine cel puţin o variabilă, a două (ii) nu este o propoziţie, iar ultima (iii) devine o propoziţie cînd fiecare din variabile sînt instanţiate. Terminologic setul obiectelor ce sînt variabile în propoziţie poartă denumirea de "universul discursului".

În structura calculului cu predicate sînt incluşi şi cuantificatori. Se scrie $\forall x$(*FUNCŢIONEAZĂ*(x)) pentru a indica faptul că orice calculator x este în funcţiune. Simbolul $\forall$ numit cuantificator universal înlocuieşte expresia

"pentru toate" sau "pentru oricare". Similar, cuantificatorul existenţial indică faptul că există cel puţin un x pentru care afirmaţia este adevărată. Cu aceste notaţii regula de mai sus poate fi transcrisă prin:

$\forall x[CALCULATOR(x) \rightarrow FUNCŢIONEAZĂ(x)]$

CALCULATOR (IMB PC)

Deci, *FUNCŢIONEAZĂ* (IMB PC) şi poate fi citită "Pentru orice x care este calculator, x este în funcţiune". Schematic structura regulii poate fi ilustrată ca în fig. 4.2.

În general expresia

$$\forall x[A(x) \rightarrow S(x)]$$

indică faptul că setul tuturor elementelor marcate prin $A(x)$ adevărate fac ca $S(x)$ să devină adevărat. În concluzie, pentru orice instanţă particulară x pentru care $T(A(x)) = 1$ sigur $T(S(x)) = 1$. Cu acestea se poate observa că un predicat se poate identifica cu o relaţie al doilea termen semnificînd un subset al setului universal de obiecte. Presupunînd ca setul u reprezintă universul discursului pentru predicatul $P(\cdot)$, şi $P(x)$ este interpretat gramatical ca "x este un predicat", atunci pentru orice x din u, $P(x)$ este fie adevărat fie fals; dacă x aparţine clasei de obiecte pentru care declaraţia "x este un predicat" este adevărată atunci

$T(P(x)) = 1$

pe cînd dacă x aparţine clasei de obiecte pentru care declaraţia "x este un predicat" este falsă, atunci:

$T(P(x)) = 0$.

În consecinţă dacă se va nota cu P_0 clasa formată, atunci

$T(P(x)) = 1$

dacă şi numai dacă $x \in P_0$.

În plus dacă $Q(\cdot)$ este un alt predicat, cu Q_0 indicînd un subset al elementelor $x \in u$ pentru care $Q(x)$ este adevărat, atunci

$$T(\forall x[P(x) \rightarrow Q(x)]) = 1$$

dacă şi numai dacă P_0 este un subset al lui $Q_0 (P_0 \subset Q_0)$.

Exemplul 1: Se consideră setul universal Y constînd din patru entităţi;

1. Un automobil în funcţionare, notat DAC.
2. Un element ce nu este în lucru, notat NOT.
3. Calculatorul IBM PC.
4. Alt calculator, numit BIT.

Se obţine deci că universul problemei este Y = {DAC, NOT, IBM PC, BIT}. Se va ilustra mai jos modul în care predicatele *CALCULATOR* () şi *FUNCŢIONEAZĂ* () corespund relaţiilor notate *CALCULATOR* şi *FUNCŢIONEAZĂ*. Dacă se presupune ca afirmaţia "toate calculatoarele funcţionează" este adevărată, adică:

$$T(\forall x[CALCULATOR\ (x) \rightarrow FUNCŢIONEAZĂ\ (x)]) = 1$$

Atunci pentru,

CALCULATOR = {IBM PC, BIT}

FUNCŢIONEAZĂ = {DAC, IBM PC, BIT}

se obţine,

T(*FUNCŢIONEAZĂ* (IBM PC)) = 1

T(*FUNCŢIONEAZĂ* (BIT)) = 1

T[*FUNCŢIONEAZĂ* (DAC)] = 1

şi

T(*FUNCŢIONEAZĂ* (NOT)) = 0

De aceea se obţine:

$T(\forall x[$*FUNCŢIONEAZĂ* $(x)]) = 0$

$T(\exists x[$*FUNCŢIONEAZĂ* $(x)]) = 1.$

Nu este necesar ca predicatele să fie unare şi chiar mai mult de un cuantificator poate fi utilizat.

Este obişnuită reprezentarea faptelor în bazele de cunoştinţe ca fişiere extinse. Un fişier de acest tip, utilizat de regulă în bazele de date relaţionale, apare ca o matrice pentru afişarea n-uplurilor în relaţiile n-are; cu alte cuvinte este un subset al produsului $E_1 \times E_2 \times \ldots \times E_n$, în care E_i este un set ce indică al i-lea obiect. Fiecare rînd al fişierului conţine un n-uplu, rîndurile fiind elemente ale relaţiei. Ca exemplu, relaţia *FUNCŢIONEAZĂ* din exemplul anterior poate fi exprimată:

IBM PC
DAC
BIT

sau în altă ordine, întrucît ordinea este irelevantă. O consecinţă este aceea ca informaţia este reprezentată în calculator în forma unor fişiere extinse. Aceasta nu necesită reprezentarea directă, ci poate fi dată în alte fişiere în conjuncţie cu regulile ce implică conective ale calculului propoziţional, predicatelor şi cuantificatorilor.

Exemplu 2: Se consideră imaginea din figura 4.3, în care se observă mai multe obiecte: o casă, un arbore, o maşină. Cinci persoane pot fi observate în această imagine: Ion şi Mihai în casă la fereastră, Ana în faţa uşii casei, Mircea în automobil, Vasile ascuns în spatele pomului. Întreaga scenă poate fi bine descrisă prin introducerea unor predicate binare STÎNGA $(\cdot, \cdot)$ şi JOS $(\cdot, \cdot)$. Dacă obiectul x este în stînga obiectului y atunci

T(STÎNGA $(x, y)) = 1$

Dacă obiectul w este mai jos decît obiectul u, atunci

T(JOS $(w, u)) = 1$

Se obţine deci

T(STÎNGA (casă, automobil)) = 1

T(STÎNGA (casă, arbore)) = 1

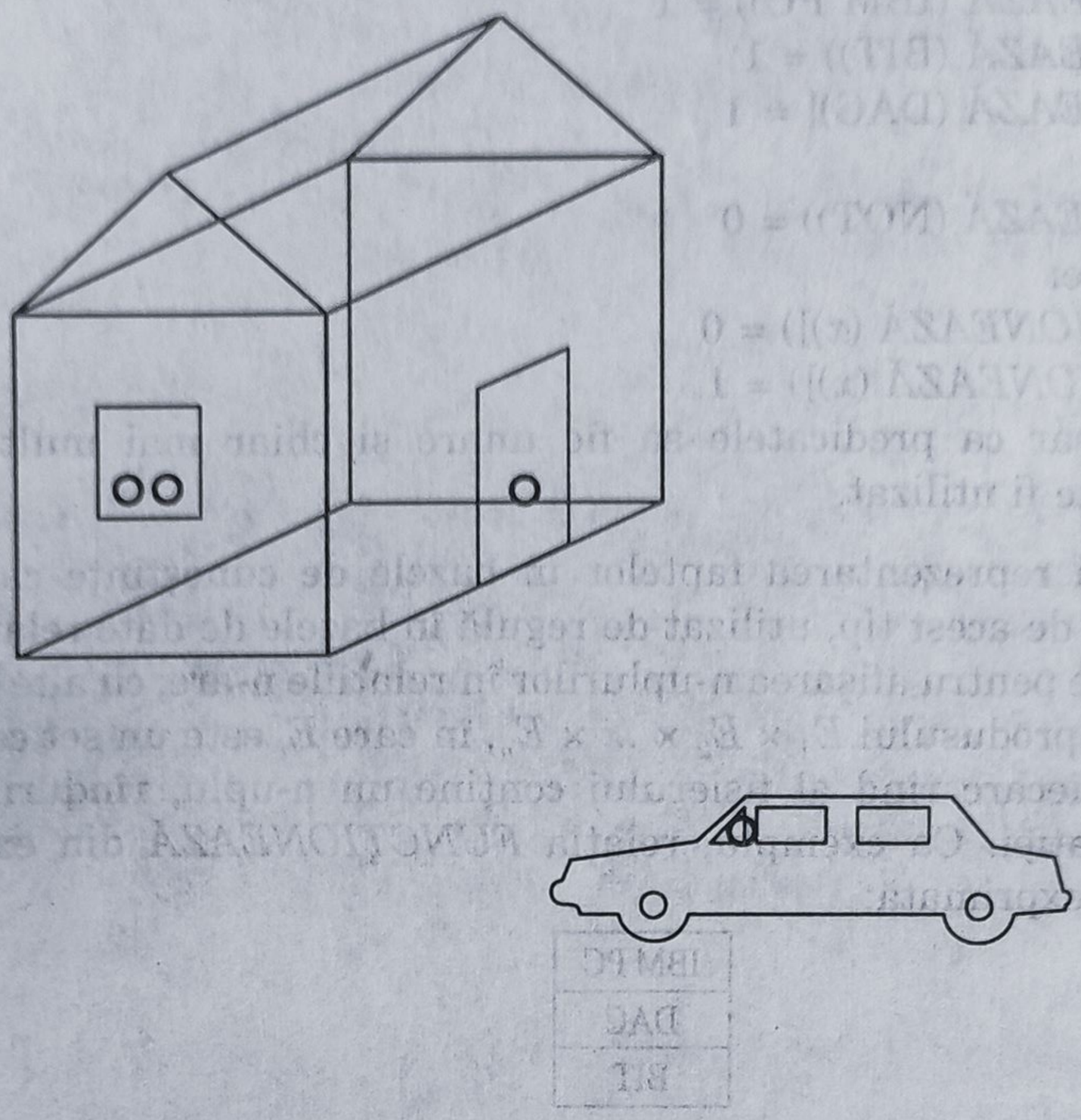
T(STÎNGA (automobil, arbore)) = 1

T(JOS (automobil, arbore)) = 1

T(JOS (automobil, casă)) = 1

T(JOS (arbore, casă)) = 1

▼ ***Fig. 4.3. Scena pentru exemplul 2***



Informaţiile precedente pot fi văzute în termeni ai relaţiilor sau în termeni ai fişierelor extinse în memoria calculatorului, prezentîndu-se numai informaţiile pentru care predicatele au valoarea "true" astfel:

STîNGA

casă	automobil
automobil	arbore
casă	arbore

JOS

automobil	arbore
automobil	casă
arbore	casă

În maniera similară se pot defini predicatele binare IN(,), ASCUNS(,),ce duc la relaţiile:

IN

Ion	casă
Mihai	casă
Mircea	automobil

ASCUNS

Vasile	arbore

Se poate uşor demonstra modul în care calculatorul determină cînd secvenţa cuantificată "Toate persoanele sînt în casă"

$\forall x$[IN (x, casa)]

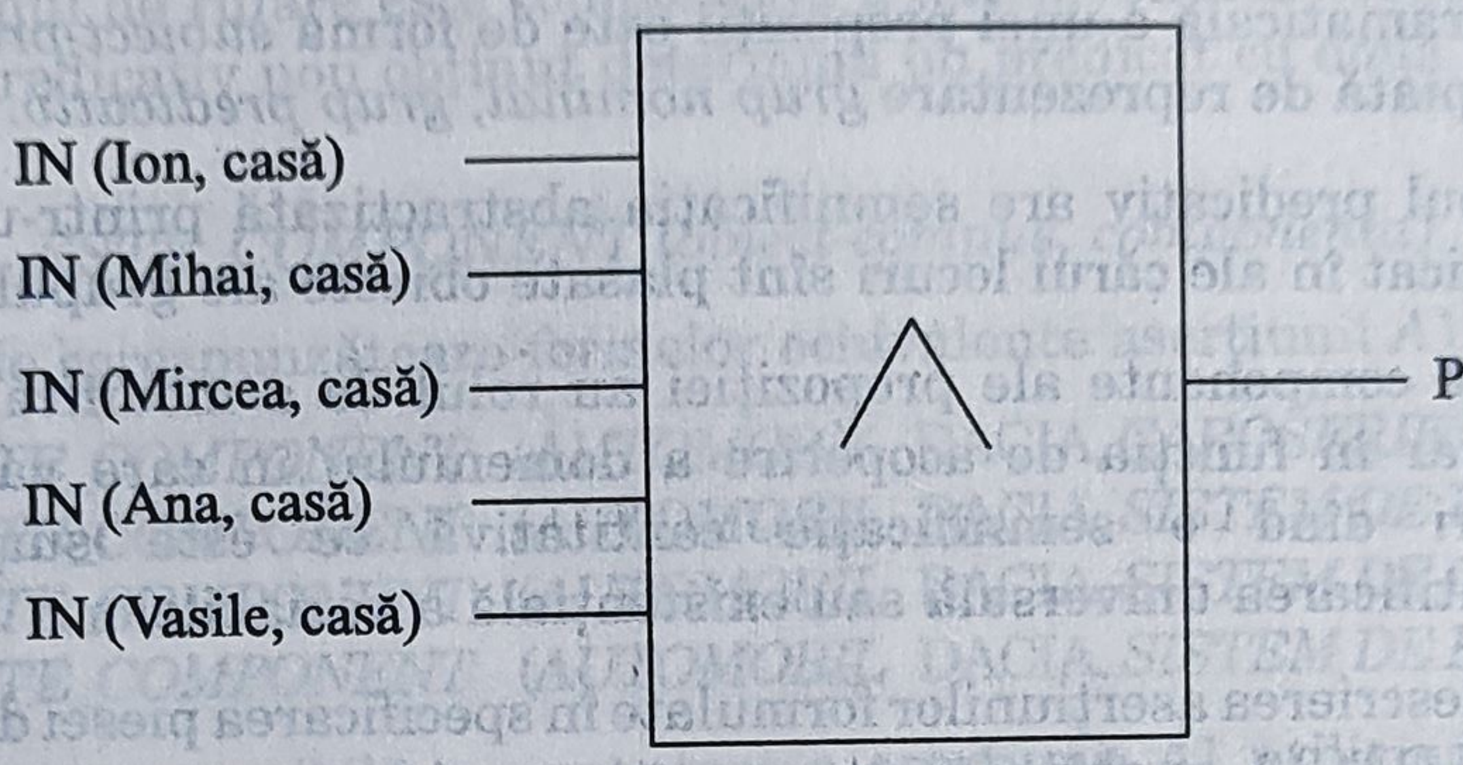
este adevărată sau falsă. Prin fixarea variabilei y = casă, IN (x, casă) este un predicat unar obţinut din IN (x, y) descris prin fişierul de mai jos.

IN (x, casă)

Ion
Mihai

Instanţierea variabilei x pentru toate persoanele produce propoziţia p definită mai jos în fig. 4.4.

▼ *Fig. 4.4. Propoziţia p*



Valoarea de adevăr pentru p este 0 întrucît cele cinci conjuncţii nu au toate valoarea adevărată. De exemplu, T(IN (Mircea, casă)) = 0, întrucît înregistrarea (Mircea, casa) nu apare în fişierul extins IN. Deci, se poate afirma că $T[\forall(x[\text{IN } (x, \text{ casa})]) = 0$. Similar valoarea de adevăr a unei cereri complexe poate fi realizată prin

$$T(\exists\, x\, \exists\, y\, \exists\, z\, \exists\, w[\text{IN } (x, y) \vee \text{JOS}(y, z) \vee \text{JOS}(w, z)] = 1$$

ce este determinată ca disjuncţia propoziţiilor corespunzătoare cu variabilele instanţiate şi determinarea faptului că cel puţin una este adevărată.

Se consideră următoarea descriere:

"Automobilul DACIA este compus din caroserie, sistem de propulsie, sistem de comandă şi sistem de rulare. Automobilul se deplasează dacă are viteza relativă diferită de zero faţă de obiectele fixe. Motorul funcţionează dacă are combustibil în rezervor şi este închis contactul de pornire".

Cele trei fraze pot fi descompuse în mai multe aserţiuni astfel:

I - *A1* - Automobilul DACIA este compus din caroserie, sistem de propulsie, sistem de comandă şi sistem de rulare.
II - *A2* - automobilul se deplasează.
- *A3* - viteza relativă faţă de obiectele fixe este diferită de zero.
III - *A4* - motorul funcţionează.
- *A5* - are combustibil în rezervor
- *A6* - este închis contactul de pornire.

Presupunînd că nu se cunosc alte fapte despre autoturism se poate transpune această cunoaştere în următoarele formule ale calculului propoziţional.

I $A1$

II $A3 \rightarrow A2$

III $(A5 \wedge A6) \rightarrow A4$

Se observă că această reprezentare nu surprinde detalii ale descrierii iniţiale întrucît acestea nu sînt propoziţii ci componente ale acestora. Propoziţiile sînt presupuse a fi bine alcătuite după reguli ce aparţin limbajului natural nu calculului propoziţional. Pentru aceasta rafinarea presupune efectuarea unei analize ce se bazează pe reprezentarea predicativă în care structura gramaticală a unei propoziţii este de forma *subiect-predicat* sau în forma apropiată de reprezentare *grup nominal, grup predicativ.*

- grupul predicativ are semnificaţia abstractizată printr-un simbol de predicat în ale cărui locuri sînt plasate obiecte ale grupului nominal;
- unele componente ale propoziţiei au rolul de a modifica valoarea de adevăr în funcţia de acoperire a domeniului în care variabilele iau valori dînd o semnificaţie cantitativă ce este surprinsă prin cuantificarea universală sau existenţială a variabilelor.

Pentru descrierea aserţiunilor formulate în specificarea piesei de cunoaştere automobilul DACIA, se obţine pentru aserţiunea $A1$ o reprezentare ce porneşte de la stabilirea unui predicat specific. Grupul predicativ al aserţiunii $A1$ determină un predicat ce va fi notat prin simbolul *ESTE_COMPUS* cu un număr de obiecte ale grupului nominal dat de numărul obiectelor din aserţiunea $A1$, adică "caroserie, sistem de propulsie, sistem de comandă, sistem de rulare" (5 obiecte în care primul este ocupat de obiectul compus, în cazul de faţă automobil).

- Aserţiunea $A1$

Astfel, în termeni de variabile formale predicatul ce generează acest tip de aserţiuni are forma generală:

ESTE_COMPUS (*obiect-compus, componenta* 1,... *componenta* 4)

în care, instaţierile de obiecte ce generează aserţiunea $A1$ fac predicatul adevărat. Se va obţine pentru aserţiunea $A1$:

ESTE_COMPUS (AUTOMOBIL DACIA, CAROSERIE, SISTEM DE PROPULSIE, SISTEM DE COMANDĂ, SISTEM DE RULARE)

Această formulare, chiar dacă are calitatea condensării întregii cunoaşteri despre automobil într-un singur predicat, are următoarele dezavantaje:

- în timpul raţionamentelor nu se utilizează decît rareori toate componentele simultan, utilizîndu-se cel mai frecvent o singură componentă;

- schimbarea numărului de locuri ale predicatelor determină redefinirea predicatului, la rafinarea sa ulterioară fiind necesară mărirea arităţii sale dacă numărul componentelor creşte. Rezolvarea acestor inconveniente poate fi făcută prin reformarea aserţiunii într-o formă logic echivalentă prin conjuncţii de propoziţii elementare al căror grup predicativ conduce la predicate binare.

Se obţine astfel pentru aserţiunea *A*1:

*A*1 (caroseria este componentă a automobilului DACIA)
(sistemul de propulsie este componentă a automobilului DACIA)
(sistemul de comandă este componentă a automobilului DACIA)
(sistemul de rulare este componentă a automobilului DACIA).

Grupul predicativ nou obţinut determină un predicat cu două locuri avînd structura:

ESTE_COMPONENT (*obiect-compus, componenta*)

Instantele corespunzătoare formelor echivalente aserţiunii *A*1 vor fi:

(1) *ESTE_COMPONENT* (*AUTOMOBIL* DACIA, *CAROSERIE*)
(2) *ESTE_COMPONENT* (*AUTOMOBIL* DACIA, *SISTEM DE PROPULSIE*)
(3) *ESTE_COMPONENT* (*AUTOMOBIL* DACIA, *SISTEM DE COMANDĂ*)
(4) *ESTE_COMPONENT* (*AUTOMOBIL* DACIA, *SISTEM DE RULARE*)

Aceasta este o posibilă reprezentare a aserţiunii *A*1 utilizînd calculul cu predicate de ordinul întîi. În această situaţie rafinarea presupune adăugarea de noi instanţe ale aceleiaşi expresii predicative cum este exemplul:

ESTE_COMPONENT (*AUTOMOBIL* DACIA, *SISTEM DE FRÎNARE*)

- Aserţiunea *A*2 - grupul predicativ este alcătuit din verbul "se deplasează", iar grupul nominal determină pentru predicatul respectiv un singur loc obţinînd forma generală:

SE DEPLASEAZĂ (*nume_obiect*)
obţinînd instanţa: *SE DEPLASEAZĂ* (*AUTOMOBIL* DACIA)

- Aserţiunea *A*3 - predicatul definit are semnificaţia "este_diferită" ce indică două locuri ce corespund la două variabile formale (*x*, *y*). Se poate folosi în acest caz predicatul

ESTE_DIFERIT (*x*, *y*)
obţinînd instanţa: *ESTE_DIFERIT* (*VITEZA_AUTOMOBIL*, 0)

- Aserţiunea *A*4 - predicatul este "funcţionează (*x*)" iar obiectul ce face adevărat acest predicat şi generează o propoziţie echivalentă cu aserţiunea *A*4 este motorul. Obţinem în acest caz reprezentarea:

FUNCŢIONEAZĂ (*nume_obiect*)
cu instanta: *FUNCŢIONEAZĂ* (*MOTOR*)

- Aserţiunea *A5* - foloseşte un predicat cu două locuri de forma generală:

  ARE (*recipient, conţinut*)
  obţinînd instanţa: *ARE* (*REZERVOR, COMBUSTIBIL*)

- Aserţiunea *A6* - utilizează un predicat cu un singur loc de forma:

  ÎNCHIS (*contact_electric*)

  obţinînd instanţa: *ÎNCHIS* (*CONTACT_PORNIRE*)

În concluzie expresiile determinate prin reprezentarea propoziţională în limbajul calculului cu predicate de ordinul întîi şi făcînd convenţia să se omită conectiva logică conjuncţie ($\wedge$) între propoziţiile elementare, respectiv păstrarea conectivei logice disjuncţie ($\vee$) în interiorul formulelor se obţine:

(1) *ESTE_COMPONENT* (*AUTOMOBIL* D1300, *CAROSERIE*)
ESTE_COMPONENT (*AUTOMOBIL* D1300, *SISTEM DE PROPULSIE*)
ESTE_COMPONENT (*AUTOMOBIL* D1300, *SISTEM DE COMANDĂ*)
ESTE_COMPONENT (*AUTOMOBIL* D1300, *SISTEM DE RULARE*)

(2) *DIFERIT* (*VITEZA_AUTOMOBIL*, 0) $\rightarrow$ *SE DEPLASEAZĂ* (*AUTOMOBIL* DACIA)

(3) (*ARE* (*REZERVOR, COMBUSTIBIL*)*ÎNCHIS* (*CONTACT_PORNIRE*)) $\rightarrow$ *FUNCŢIONEAZĂ* (*MOTOR*)

În acest exemplu de reprezentare predicatele exprimă obiecte bine precizate pentru obiectul unic DACIA. În situaţia în care se reprezintă conceptul generic automobil ce se instanţează în obiecte individuale ca D1300, FIAT,... etc. se poate spune că a fi automobil este o caracteristică a obiectului respectiv ce se exprimă printr-un predicat, ca de exemplu *AUTOMOBIL* $\leftarrow (x)$. Predicatul va fi adevărat pentru orice obiect individual ce este în realitate automobil. Cum numărul de automobile individuale este foarte mare, a scrie pentru fiecare că are caroserie, sistem de propulsie... etc. este neconvenabil, motiv pentru care se vor utiliza cuantificatori universali.

Astfel utilizarea cuantificatorilor universali în cazul tipic determinat mai sus conduce la:

$(\forall)\ x$ *AUTOMOBIL* $(x) \rightarrow$ *ARE* (x, *CAROSERIE*)
$(\forall)\ x$ *AUTOMOBIL* $(x) \rightarrow$ *ARE* (x, *SISTEM DE PROPULSIE*)
$(\forall)\ x$ *AUTOMOBIL* $(x) \rightarrow$ *ARE* (x, *SISTEM DE COMANDĂ*)
$(\forall)\ x$ *AUTOMOBIL* $(x) \rightarrow$ *ARE* (x, *SISTEM DE RULARE*)

În acest caz se poate considera că toate obiectele fizice menţionate sînt concepte generice nu individuale. Pentru aceasta se vor înlocui conceptele generice: *CAROSERIE, SISTEM DE PROPULSIE*, etc. cu variabile simbolice, iar caracteristica se specifică prin predicate cum s-a procedat la conceptul de automobil.

$\forall\ x$ *CAROSERIE* $(x) \rightarrow$ *ESTE ELEMENT* (x, C)

în care, C este o mulţime de descrieri ale caroseriilor, diferite după formă şi caracteristici ce se încadrează în conceptul generic *CAROSERIE.*

Predicatul *ESTE ELEMENT* semnifică apartenenţa obiectului ataşat variabilei formale x la mulţimea C a caroseriilor. Problemele de apartenenţă la o mulţime pentru entităţi ce sînt termeni ai altor predicate pot fi rezolvate prin folosirea predicatelor dar conduc prin substituire la predicate de ordin superior. Aceste inconveniente pot fi rezolvate prin utilizarea ca termeni ai predicatelor a simbolurilor funcţionale. Se poate scrie astfel:

$$CAROSERIE : X \rightarrow C$$

văzută ca o aplicaţie de la mulţimea X a automobilelor la mulţimea C a caroseriilor situaţie în care termenul $CAROSERIE(x)$ are ca valoare un simbol identificator de caroserie din mulţimea C a acestora. Se obţine astfel:

$$\forall\, x\ AUTOMOBIL(x) \vdash\ ARE(x,\ CAROSERIE(x))$$

ce este o expresie bine formată în calculul cu predicate de ordinul întîi. În situaţia cînd nu toate elementele unei mulţimi satisfac o formulă, însă există cel puţin un element care o satisface se utilizează cuantificatorul existenţial al variabilei formale ce se ataşează elementului.

Astfel pentru a reprezenta aserţiunea:
"Cineva merge la piaţă"
se poate folosi formula: $(\exists)\ x\ OM(x) \wedge MERGE\ (x,\ PIAŢĂ)$.

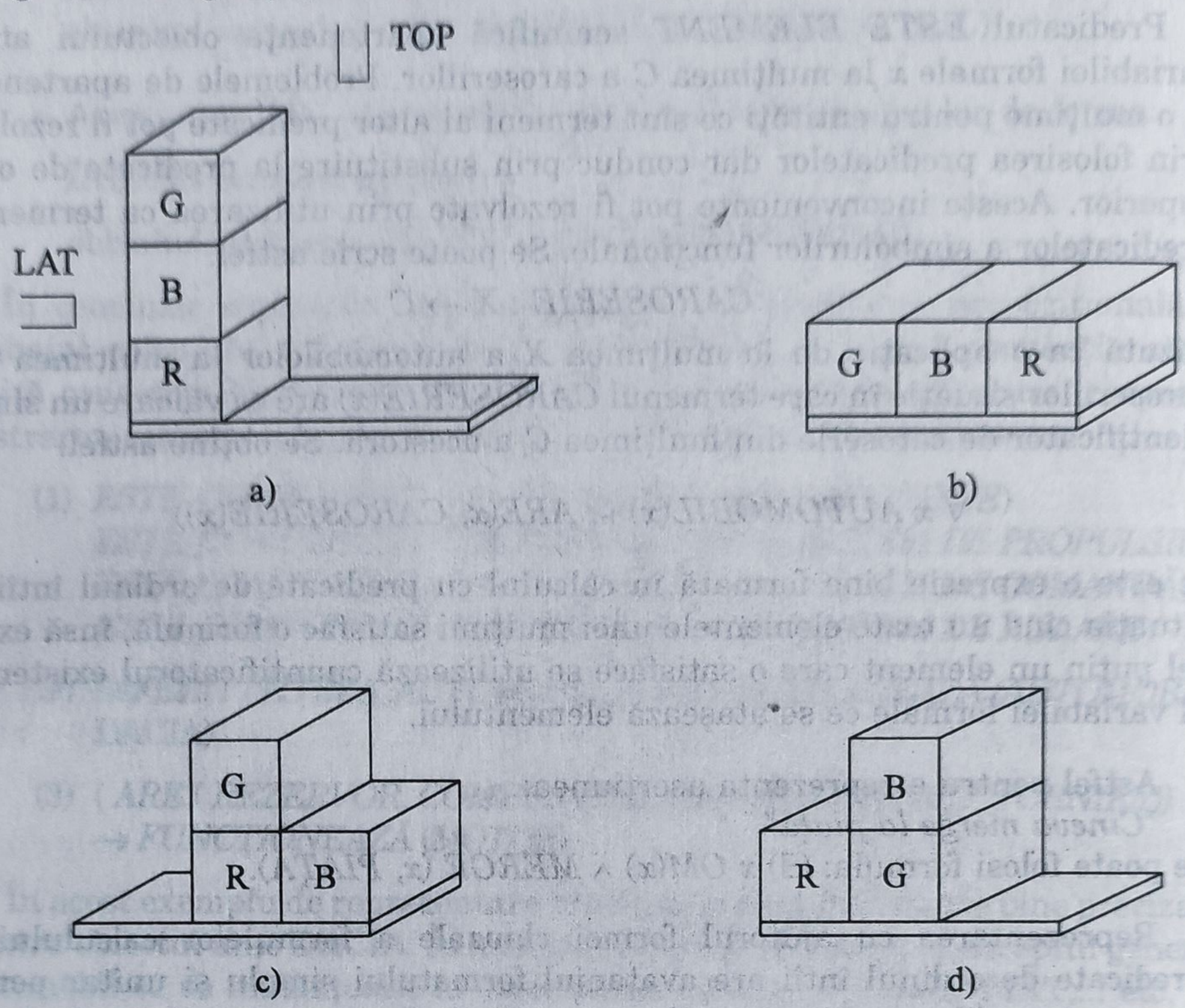
Reprezentarea cu ajutorul formei clauzale a formulelor calculului cu predicate de ordinul întîi are avatanjul formatului simplu şi unitar pentru toate categoriile constituente ale cunoaşterii. Se poate cu ajutorul clauzei exprima într-o formulă compactă caracteristicile automobilului formulate mai sus.

$$\begin{aligned}\forall\, x\ & ARE\ (x,\ CAROSERIE) \wedge \\ & ARE\ (x,\ SISTEM\ DE\ PROPULSIE) \wedge \\ & ARE\ (x,\ SISTEM\ DE\ COMANDĂ) \wedge \\ & ARE\ (x,\ SISTEM\ DE\ RULARE) \rightarrow AUTOMOBIL(x)\end{aligned}$$

Un exemplu tipic de utilizare a calculului cu predicate este cazul controlului unui robot. Pentru a ilustra modul în care un robot înţelege informaţiile percepute utilizînd calculul cu predicate, se consideră lumea exterioară ca fiind formată dintr-o masă pe care sînt dispuse trei blocuri diferit colorate. Problema care se pune este relativă la modul de rearanjare a celor trei blocuri ştiind că pot fi aşezate toate cele trei blocuri pe masă sau pot fi aşezate în stivă unul peste altul, sau orice combinaţie a celor două aşezări.

Se presupune că robotul a fost programat ca posibila configuraţie a blocurilor să fie cunoscută prin colectarea informaţiilor referitoare la poziţie de la doi senzori, informaţii ce specifică culoarea blocului din faţa sa. Un senzor percepe informaţiile referitoare la aşezarea pe verticală, senzor care este aşezat lateral şi celălalt informaţia referitoare la aşezarea pe orizontală,

▼ ***Fig. 4.5. Configuraţii posibile ale blocurilor***



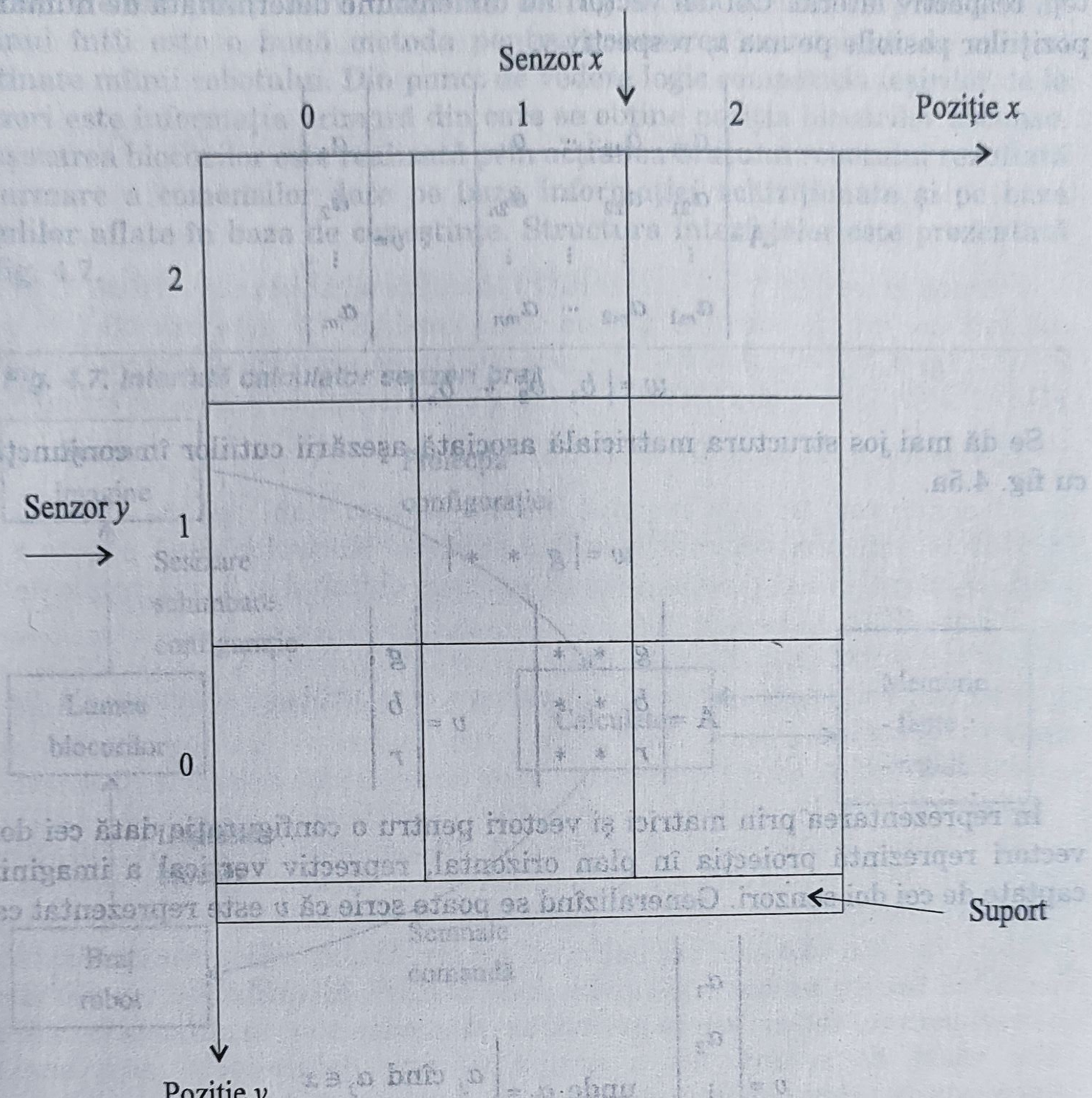
Imaginînd un sistem de coordonate cartezian, cele 9 poziţii posibile ale blocurilor pot fi reprezentate printr-o serie de predicate ce descriu poziţiile relative într-o grilă ca cea din fig. 4.6.

Se vor introduce predicatele LOC(z, i, j) ceea ce indică faptul că blocul de culoare z este poziţionat la locaţia (i, j) a sistemului de coordonate. Întregul i indică locaţia pe axa x ce este determinată prin senzorul TOP, întregul j locaţia pe axa y determinată de senzorul LAT. Predicatul TOP(z, i) va fi utilizat pentru reprezentarea ieşirii senzorului superior, în care i reprezintă poziţia corespunzătoare pe axa x. Analog senzorul lateral indică poziţia blocului de culoare z pe axa y. Pentru aceasta predicatul LAT(z, j) a fost introdus.

Pentru configuraţia ilustrată în fig. 4.5a se pot scrie valorile de adevăr ale predicatelor ca mai jos.

T(LOC(g, 0, 2)) = 1
T(LOC(b, 0, 1)) = 1
T(LOC(r, 0, 0)) = 1
T(LAT(g, 2)) = 1
T(LAT(b, 1)) = 1
T(LAT(r, 0)) = 1

▼ *Fig. 4.6. Coordonate blocuri*



$T(\text{TOP}(g, 0)) = 1$

Cu aceste informaţii pot fi scrise relaţiile

$\text{LOC} = ((g, 0, 2), (b, 0, 1), (r, 0, 0))$
$\text{LAT} = ((g, 2), (b, 1), (r, 0))$
$\text{TOP} = ((g, 0))$

corespunzătoare localizării poziţiei pe axele x şi y a celor trei blocuri.

Reprezentarea obiectelor poate fi specificată prin matrici sau vectori. O matrice de acest tip este de dimensiune mxn în care elementele aparţin setului de elemente X, absenţa unui element într-o poziţie fiind marcată prin *.

Elementul a_{11} indică localizarea primului rînd şi a primei coloane în sistemul de coordonate. Vectorii linie şi coloană reprezintă poziţia indicată de senzorii top, respectiv lateral. Cei doi vectori au dimensiune determinată de numărul poziţiilor posibile pe axa x, respectiv y.

$$A = \begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{vmatrix} \qquad v = \begin{vmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{vmatrix}$$

$$w = | \; b_1 \;\; b_2 \;\; \cdots \;\; b_n \; |$$

Se dă mai jos structura matricială asociată aşezării cutiilor în conjuncţie cu fig. 4.5a.

$$w = | \; g \;\; * \;\; * \; |$$

$$A = \begin{vmatrix} g & * & * \\ b & * & * \\ r & * & * \end{vmatrix} \qquad v = \begin{vmatrix} g \\ b \\ r \end{vmatrix}$$

În reprezentarea prin matrici şi vectori pentru o configuraţie dată cei doi vectori reprezintă proiecţia în plan orizontal, reprectiv vertical a imaginii captate de cei doi senzori. Generalizînd se poate scrie că v este reprezentat ca

$$v = \begin{vmatrix} a_1 \\ a_2 \\ \vdots \\ a_m \end{vmatrix} \quad \text{unde } a_i = \begin{vmatrix} a_j \text{ cînd } a_j \in x \\ * \text{ altfel} \end{vmatrix}$$

Similar proiecţia pe axa y determină vectorul

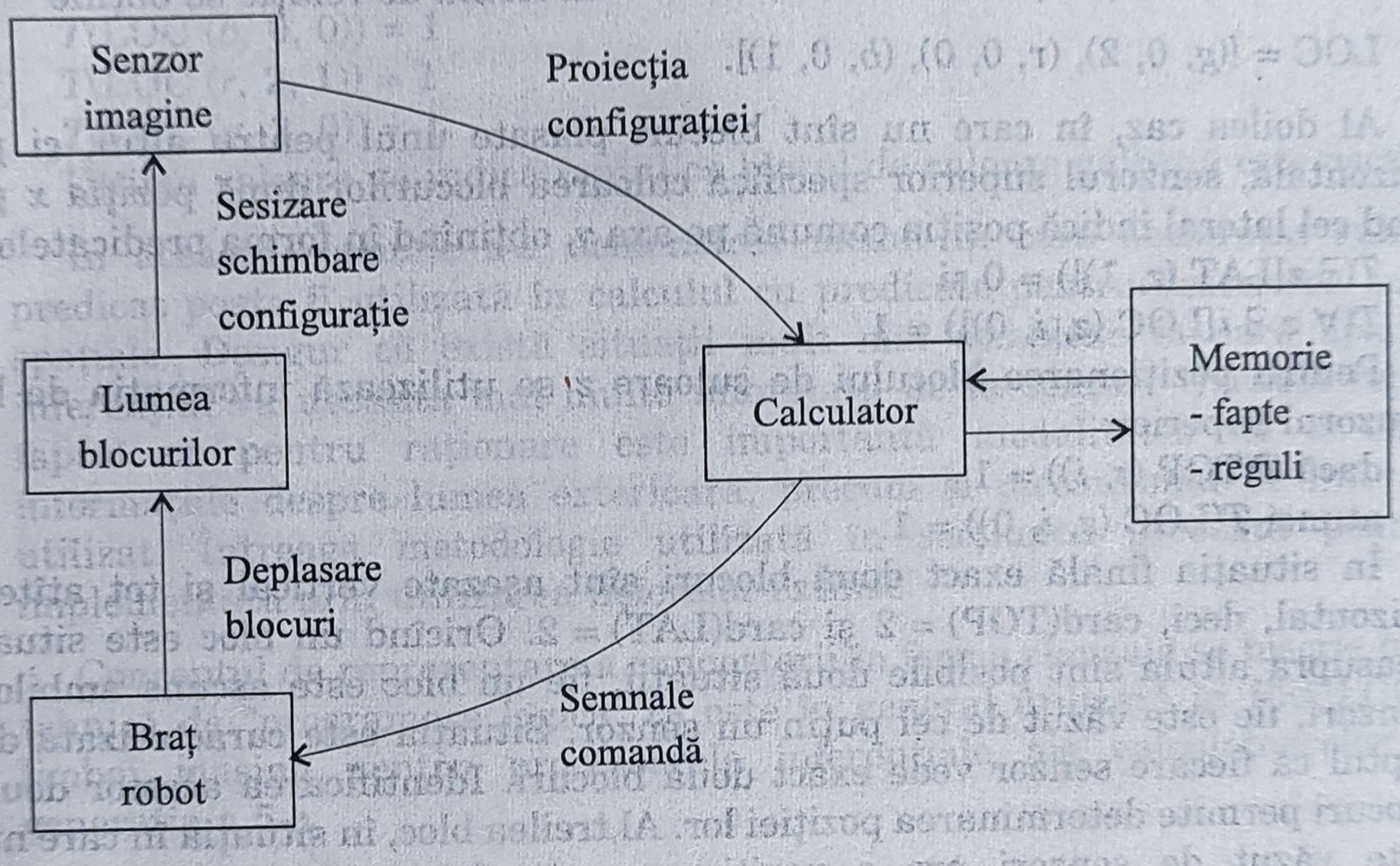
$$w = | \; b_1 \;\; b_2 \;\; \ldots \;\; b_n \; |$$

$$\text{unde } b_i = \begin{vmatrix} b_j \text{ cînd } b_j \in x \\ * \text{ altfel} \end{vmatrix} .$$

O problemă esenţială este găsirea matricii A atunci cînd se cunosc vectorii proiecţiei $P_x(A) = v$ şi $P_y(A) = w$. În general problema nu are soluţie. Totuşi, în multe cazuri pornind de la cunoştinţele despre poziţia blocurilor aferent informaţiei obţinute de la senzori se poate obţine o unică matrice reprezentînd

imaginea blocurilor. În alte cazuri datorită faptului că un bloc este ascuns de altele nu este posibilă determinarea poziţiei exacte a acestora. Un astfel de caz este cel aferent poziţionării blocurilor din fig. 4.5d. Calculul cu predicate de ordinul întîi este o bună metoda pentru generarea semnalelor de control destinate mîinii robotului. Din punct de vedere logic compoziţia ieşirilor de la senzori este informaţia primară din care se obţine poziţia blocurilor ascunse. Reaşezarea blocurilor este realizată prin acţiunea braţului robotului rezultată ca urmare a comenzilor date pe baza informaţiei achiziţionate şi pe baza regulilor aflate în baza de cunoştinţe. Structura interfeţelor este prezentată în fig. 4.7.

▼ ***Fig. 4.7. Interfaţă calculator senzori braţ***



Braţul robotului reconfigurează poziţia blocurilor, senzorii determină schimbarea configuraţiei, iar calculatorul prin intermediul informaţiei apriorice conţinute în regulile logice determină poziţia finală a noii configuraţii a blocurilor. Cînd configuraţia nu este cea dorită, semnale de control sînt transmise către braţul robotului pentru a determina acesta să schimbe poziţia blocurilor către configuraţia dorită.

Se consideră cazul în care blocurile sînt aşezate vertical unul peste altul. În această situaţie senzorul TOP va furniza locaţia comună x a tuturor blocurilor, senzorul lateral indică culoarea fiecărui bloc în conjuncţie cu poziţia lor pe axa y. În TOP apare numai un element, fapt pentru care card(TOP) = 1 şi card(LAT) = 3 (prin card () s-a notat numărul elementelor mulţimii diferite de *). Situaţia este în acest moment caracterizată prin

$$T(\exists z[\mathrm{LAT}(z, 2)]) = 1.$$

În plus T(LOC (z, i, 2)) = 1, unde locaţia orizontală i a blocului din vîrful stivei de culoarea z este determinată prin T(TOP(z, i)) = 1. Mai mult, alte două blocuri au aceeaşi poziţie orizontală i şi aşezarea lor este detectată de senzorul lateral. Dacă T(LAT (u, 1)) = 1, atunci T(LOC (u, i, 1)) = 1, respectiv dacă T(LAT (w, 0)) = 1, atunci T(LOC (w, i, 0)) = 1.

În situaţia ilustrată în fig. 4.5a se obţine pentru ieşirile senzorilor

LAT = [(g, 2), (b, 1), (r, 0)]
TOP = [(g, 0)]

Întrucît T(LAT (g, 2)) = 1, rezultă că blocurile sînt aşezate vertical în prima coloană cu cel de culoare galben sus, condiţie ce este specificată prin T(LOC, (g, 0, 2)) = 1. Mai departe prin utilizarea senzorului lateral se obţine T(LOC (b, 0, 1)) = 1 şi T(LOC (r, 0, 0)) = 1. În formă de relaţie se obţine

LOC = [(g, 0, 2), (r, 0, 0), (b, 0, 1)].

Al doilea caz, în care nu sînt blocuri plasate unul pentru altul, ci pe orizontală, senzorul superior specifică culoarea blocurilor după poziţia x pe cînd cel lateral indică poziţia comună pe axa y, obţinînd în forma predicatelor:

$T(\exists z[\text{LAT}(z, 1)]) = 0$ şi

$T(\forall z \exists i[\text{LOC}(z, i, 0)]) = 1$.

Pentru poziţionarea blocului de culoare z se utilizează informaţia de la senzorul superior:

dacă T(TOP (z, i)) = 1

atunci T(LOC (z, i, 0)) = 1.

În situaţia finală exact două blocuri sînt aşezate vertical şi tot atîtea orizontal, deci, card(TOP) = 2 şi card(LAT) = 2. Oricînd un bloc este situat deasupra altuia sînt posibile două situaţii, fie un bloc este ascuns ambilor senzori, fie este văzut de cel puţin un senzor. Situaţia este caracterizată de faptul că fiecare senzor vede exact două blocuri. Identificarea acestor două blocuri permite determinarea poziţiei lor. Al treilea bloc, în situaţia în care nu este văzut de senzori are o poziţie ce este determinată prin analiza informaţiilor primite de la cei doi senzori. În termenii calculului cu predicate se obţine:

$$T(\exists z \exists i[\text{TOP}(z, i) \wedge \text{LAT}(z, 0)]) = 1$$

şi localizarea unui singur bloc de culoarea z este imediat găsită din informaţiile ce provin de la cei doi senzori.

T(LOC (z, i, 0)) = 1.

Blocul de culoare u este deasupra şi este observat de către cei doi senzori direct

T(LOC (u, j, 1)) = 1.

În final, întrucît $z \# u$ şi ambele sînt elemente ale lui $X = \{r, g, b\}$, blocul ascuns este elementul w, a cărui locaţie va fi specificată prin

T(LOC (w, j, 0)) = 1.

Ultima situaţie corespunde cazului în care ambii senzori detectează două blocuri şi numai unul dintre acestea este comun. Un bloc este deasupra altui

bloc şi are o localizare obţinută direct de la senzori. Un alt bloc văzut prin senzorul lateral are o localizare orizontală identică cu cea a blocului superior. În final, al treilea bloc trebuie să fie aşezat pe masă şi are o localizare orizontală dată de senzorul vertical. Utilizînd predicatele, blocul superior de culoare z are nivelul 1 şi T(TOP (z, i)) = 1. Deci, T(LOC $(z, i, 1)$) = 1. Blocul de jos de culoare u este văzut de senzorul lateral T(LAT $(u, 0)$) = 1 şi T(LOC $(u, i, 0)$) = 1. Ultimul bloc are culoare $w \in X$ - $\{z, u\}$ dată de senzorul superior şi T(TOP (w, j)) = 1, implicînd ca T(LOC $(w, j, 0)$) = 1.

Exemplu:

Considerînd că ieşirile celor doi senzori sînt date de relaţiile:

TOP = $\{(b, 0), (r, 2)\}$ şi

LAT = $\{(b, 0), (r, 1)\}$

card(TOP) = 2 şi card(LAT) = 2, corespunde situaţiei 3. Se va obţine

T(LOC $(b, 0, 0)$) = 1

T(LOC $(r, 2, 1)$) = 1

T(LOC $(g, 2, 0)$) = 1.

Ultima valoare va indica faptul că blocul de culoare galbenă este ascuns.

În această secţiune s-a demonstrat modul în care declaraţia subiect-predicat poate fi utilizată în calculul cu predicate pentru descrierea scenei spaţiale. Desigur că există situaţii mult mai complexe în care aparatul inferenţial va necesita mai multe informaţii de la senzori. Trebuie reţinut faptul că pentru raţionare este importantă modalitatea de a obţine informaţiile despre lumea exterioară, precum şi mecanismul de inferenţă utilizat. Întreaga metodologie utilizată în secţiunea prezentă poate fi implementată prin utilizarea fişierelor extinse.

Conceptul de reprezentare a cunoaşterii în forma clauzală se înscrie într-o tehnică de "programare logică" ce este în general utilizată în PROLOG ca limbaj maşina pentru procesoarele inferenţiale ale calculatoarelor din generaţia a 5-a.

4.2.2. Metode procedurale de reprezentarea cunoaşterii

Reprezentarea procedurală nu este legată de cercetările în domeniul inteligenţei artificiale. Programele de calcul cunoscute sînt piese de cunoaştere obţinute prin reprezentarea procedurală a unor algoritmi. Reprezentarea cunoaşterii asupra acestor entităţi utilizînd limbajul calculului cu predicate de ordinul întîi are dificultăţi datorate caracterului declarativ al metodei, caracter ce nu corespunde cu natura procedurală a pieselor de cunoaştere. Reprezentarea declarativă implică reprezentări de multe ori artificiale pentru implementarea conceptului de control. Reprezentarea procedurală nu poate evita complet caracterul declarativ al pieselor de cunoaştere despre obiecte şi fapte. Este motivul pentru care la ora actuală prin îmbinarea avantajelor celor două tipuri de reprezentări remarcabile. Metodele declarative sînt acele metode ce oferă facilităţi pentru specificarea aspectelor statice ale cunoaşterii, pentru specificarea componentelor, proprietăţilor, evenimentelor, stărilor.

Metodele procedurale - se bazează pe aspectele dinamice ale cunoaşterii asupra modului de folosire a pieselor de cunoaştere pentru efectuarea inferenţelor, asupra determinării de noi fapte prin executarea unor noi operaţii asupra pieselor de cunoaştere. Metodele procedurale presupun utilizarea unor simboluri prin care se identifică proceduri ce sînt evaluate de procesoarele interpretative ale limbajelor de nivel înalt. Un exemplu de acest tip este limbajul LISP. Reprezentarea procedurală este realizată în termenii unor simboluri prin care cunoaşterea este utilizată la rezolvarea problemelor. Astfel, fie piesa de cunoaştere ornitorinc ce poate fi reprezentată procedural prin forma condensată următoare:

PENTRU_A_STABILI ORNITORINC (*X*);
DECLARĂ_PREDICATE (*X*);
DOVEDEŞTE MAMIFER (*X*);
DOVEDEŞTE DEPUNE_OUĂ (*X*);
DOVEDEŞTE ÎNOATĂ (*X*).

În această piesă de cunoaştere simbolurile *PENTRU_A_STABILI, DECLARĂ_PREDICATE* şi *DOVEDEŞTE* sînt identificatori de proceduri prin care se urmăreşte:

- determinarea apartenenţei unui obiect la o clasă;
- înscrierea proprietăţilor obiectului *x* în forma de predicate;
- determinarea valorii unui obiect al unei piese de cunoaştere.

Prima procedură le foloseşte pe celelalte drept componente în corpul reprezentării procedurale a cunoaşterii. Ordinea de parcurgere este cea specificată efectul fiind că pentru a arăta ca *X* este ornitorinc este necesar mai întîi să se descrie piesa de cunoaştere *X* prin intermediul predicatelor ce îi stabileşte proprietăţile, apoi se apelează procedura de determinare a valorii atributelor lui *X*. În schimb dacă se dau faptele că *X* este mamifer, depune ouă şi că înoată, acestea pot servi la determinarea speciei de mamifer.

Desigur că reprezentarea procedurală este dependentă de tipul problemei ce se rezolvă cu piesele de cunoaştere reprezentate. În măsura în care cunoaşterea devine tot mai complexă, piesele de cunoaştere devin tot mai numeroase. Metodele procedurale au ca principal avantaj uşurinţa aplicării regulilor existente, specifice domeniului de competenţă aplicativă. Cunoaşterea astfel reprezentată este directă şi conduce la performanţe de timp mult mai bune. Are însă dezavantajul major al flexibilităţii scăzute în utilizarea pieselor de cunoaştere pentru efectuarea altor inferenţe decît cele specificate explicit prin procedurile aferente.

4.2.3. Reprezentarea cunoaşterii prin reţele de producţie

Acest mod de reprezentare a cunoaşterii este una dintre cele mai utilizate în sisteme expert. Metoda se bazează pe separarea componentelor obişnuite ale calculului, în scopul manipulării uşoare în procesele la care sînt utilizate.

Cunoaşterea în reţelele de producţie este de natură procedurală şi pot fi definite următoarele componente:

- *cunoaştere declarativă sau factuală* ce reprezintă piese de cunoaştere stocate sub forma unor structuri de date într-o colecţie numită şi *context;*
- *cunoaştere procedurală* reprezentată sub forma unei colecţii de tip condiţie-acţiune numite şi *reguli de producţie,* colecţie ce formează *baza de reguli;*
- *cunoaşterea strategică sau de control* formată din reguli ce privesc secvenţele de acţiuni în procesul de rezolvare.

Sistemul construit în jurul regulilor de producţie se bazează pe structura specifică compusă din cele două părţi <partea de premisă> → <partea de acţiune>. Într-o altă exprimare un sistem de producţie este compus dintr-o bază de date şi un set de reguli. Condiţiile unei reguli pot fi considerate ca o bază de date, ce returnează un indicator de succes sau eroare. Concluzia unei reguli este o acţiune ce manipulează date din baza de date, şi în plus un control al sistemului determină secvenţa regulilor utilizate.

În descriere generală se poate menţiona faptul că sistemele de producţie sînt similare gramaticilor şi apar chiar în definirea gramaticilor Chomsky. Astfel $G = (V, \Sigma, P, S)$ este o gramatică de structură a frazei, unde:

- V reprezintă un set finit de simboluri numit adesea şi *alfabet total;*
- $\Sigma \subseteq V$ reprezintă un set finit de simbluri ale alfabetului numite *simboluri terminale* sau simplu *terminale;*
- P o submulţime a produsului cartezian $(V - \Sigma - s) * V$. Elementele lui P, deci perechile ordonate (u, w), se mai scriu în mod obişnuit sub forma $u \rightarrow w$ şi sînt numite *producţii* sau reguli de rescriere;
- $\{s\}$ un simbol iniţial sau simbol de start.

Se porneşte de la ideea că un sistem de producţii este compus dintr-o bază de date şi un set de reguli. Condiţiile unei reguli pot fi văzute încă similare cu o bază de date ce întoarce un indicator de succes sau eroare. Concluzia unei reguli este o acţiune ce manipulează baza de date şi în plus un control va determina secvenţa regulilor utilizate. Baza de date conţine cuvinte construite cu simboluri din V. În concluzie, un sistem de producţie R poate fi considerat un dublet $R = (D, P)$, în care:

D este o bază de date

P set finit de reguli.

Baza de date este constituită dintr-un set de termeni, iar o regulă are forma generală IF c THEN t, în care condiţia c este constituită din termeni, paranteze, conective $\vee$, $\wedge$, $\neg$, iar concluzia t este formată dintr-un singur termen. Termenii sînt constituiţi din parametrii $x \in \mathcal{P}(R)$ şi din valori $a \in \mathbb{V}(R)$. Se presupune că $\mathcal{P}(R)$ şi $\mathbb{V}(R)$ sînt mulţimi finite şi $\mathcal{P}(R) \cap \mathbb{V}(R) = \varnothing$.

Sintaxa $x = a$, respectiv $x \# a$ cu $x \in P(R)$ şi $a \in \mathbb{V}(R)$ va trebui interpretată că x este mărginit de a şi, respectiv că x nu este mărginit de a. Un termen t este adevărat pentru baza de date D dacă şi numai dacă $t \mid D$. Succesul sau eşecul unui termen arbitrar în condiţia c este redus la succesul sau eşecul termenilor componenţi.

Observaţii. 1. În general nu se permite o conjuncţie de termeni la o concluzie. În acest sens o regulă de tipul

IF c THEN t & s

se va scrie ca două reguli în forma

IF c THEN t

IF c THEN s

2. Definiţia implică faptul că $\neg x = a$ nu este aceeaşi cu $x \neq a$ întrucît $x = a \in D$ nu este acelaşi cu $x \neq a \in D$. *Exemplu:* Fie R un sistem de producţii cu:

$R = (\{x = a, y = b, z = a\}$

$\{\text{IF } (x = a \wedge \neg y = c \wedge z \neq a) \text{ THEN } q \neq d\})$.

Întrucît la condiţia regulii, baza de date returnează succes se va obţine în baza de date $D' = \{x = a, y = b, z \neq a, q \neq d\}$.

Se poate afirma că structura generală a unei reguli de producţie este următoarea:

<partea condiţie> → *<partea acţiune>*

avînd interpretarea,

IF <partea condiţie> este îndeplinită,

THEN se execută <partea acţiune>

sau în forma mai generală:

IF <partea condiţie> este îndeplinită,

THEN se execută <partea acţiune_1>

ELSE se execută <partea acţiune_2>.

Dacă într-o regulă condiţia este satisfăcută se spune că regula este selectată pentru declanşare sau regula este aplicabilă. Regulile aplicabile intră într-o mulţime a regulilor aplicabile, mulţime din care este selectată regula cu cea mai mare prioritate după criterii ce se vor prezenta. Mecanismul regulilor de producţie este împrumutat din teoria limbajelor formale, este de natură procedurală şi poate cel mai apropiat de modul de realizare a programelor clasice. Însăşi structura regulii de tipul IF - THEN - ELSE este una din structurile de control întîlnită în toate limbajele de nivel înalt.

Mecanismul interpretativ al regulilor de producţie conţine următorii paşi:

- selectarea tuturor regulilor ce conţin piese de cunoaştere ce satisfac partea de condiţie numită şi *corespondenţă*. Mulţimea acestor reguli formează *mulţimea condidată*, denumire ce are originea în faptul că elementele acestei mulţimi intră într-o competiţie în urma căreia se decide care regulă este efectiv aplicată;
- rezolvarea conflictelor prin care din mulţimea regulilor aplicabile se elimină mai întîi regulile ce duc la aceleaşi rezultate, după care în conjuncţie cu mecanismul de asertare a priorităţii pentru problema respectivă se selectează regula ce va fi efectiv aplicată;

- ♦ execuţia părţii *acţiune* a regulii cu cea mai mare prioritate, în situaţia în care sînt producţii aplicabile;
- ♦ pornind de la contextul modificat în urma aplicării regulilor anterioare se reia ciclul începînd cu faza de corespondenţă, atîta timp cît ciclul produce acţiuni materializate prin modificarea contextului.

Oprirea mecanismului interpretativ poate avea loc dacă acţiunea unei producţii specifică concret oprirea, sau se selectează o producţie vidă.

Definiţie: *Un sistem de producţii este un cvintuplu* $SP = (K, P, \sigma, \varphi, p_2)$, *în care,*

K reprezintă contextul ca o mulţime de piese de cunoaştere factuală recunoscute de celelalte componente ale sistemului;

P baza de reguli ca mulţime finită de reguli de producţie;

σ *funcţia succesor la îndeplinirea cu succes a condiţiei, funcţie definită pe mulţimea P cu valori în mulţimea* $P \cup \{\lambda\}$;

φ *funcţia succesor în caz de eşec a condiţiei ca o aplicaţie între aceleaşi mulţimi ca şi funcţia succesor la succes;*

p_s *regula de producţie iniţială de la care porneşte procesul de selectare a regulilor.*

S-a notat cu λ producţia vidă.

Întrucît, partea condiţie şi partea acţiune sînt forme complexe ce conţin mai multe condiţii şi mai multe acţiuni forma cea mai generală a unei reguli de producţie este:

$$p = (c_1 \wedge c_2 \wedge \ldots \wedge c_i) \rightarrow (a_1; a_2; \ldots a_p)$$

în care c_j, $1 <= j <= i$ sînt condiţiile şi a_k, $1 <= k <= p$ sînt acţiunile.

Pentru ca o producţie p să fie selectată, ţinînd cont de conectiva logică dintre condiţiile c_j, este necesar ca în context să existe piese de cunoaştere ce satisfac toate aceste condiţii. Dacă regula este selectată atunci acţiunile a_k $1 <= k <= p$ sînt executate.

Controlul sistemelor de producţii este specificat complet de cele trei componente ale sale σ, φ şi p_s şi deci se pot furniza principalele strategii de control. O metodă des întîlnită este cea prin care se concepe o *strategie exhaustivă* ce selectează toate regulile ce se potrivesc condiţiei date de context. O astfel de strategie este indicată mai jos.

Fie contextul $K = \{k_1, k_2, \ldots, k_n\}$
şi producţiile $P = \{p_1, p_2, \ldots, p_m\}$

$$\begin{cases} \sigma(p_i) = \varphi(p_i) = p_{i+1}, \text{ pentru } 1 <= i <= m - 1 \\ \sigma(p_n) = \varphi(p_n) = p^n \end{cases}$$

dacă, mulţimea candidată este diferită de mulţimea vidă şi funcţia de rezlvare a conflictelor de prioritate c(MC) dă rezultatul p.

$$\sigma(p_n) = \varphi(p_n) = \lambda$$

dacă, mulţimea candidată este vidă sau funcţia de rezolvare a conflictelor oferă rezultatul STOP.

$$p_s = p_1$$

Altă metodă de strategie de control este inspirată din algoritmii normali Markov:

$K = \{k_1, k_2, ..., k_n\}$

$P = \{p_1, p_2, ..., p_m\}$

$\sigma(p_i) = p_1$ pentru $1 <= i <= n$

$\varphi(p_i) = p_{i+1}$ pentru $1 <= i <= n - 1$

$\varphi(p_n) = \lambda$

$p_s = p_1$.

Conform acestei strategii sistemul cercetează condiţiile din context începînd cu prima regulă, iar la întîlnirea primei reguli aplicabile o aplică după care comută controlul de regulă p_1 cu noul context. În situaţia de succes controlul este comutat la regula p_{i+1} realizînd astfel o explorare succesivă. În condiţiile în care nici o regulă nu este aplicabilă se selectează conform funcţiei succesor la eşec producţia vidă.

O metodă des întîlnită este şi cea utilizată în sistemul expert EXSYS, prin care se alege din mulţimea candidată producţia cu cel mai mare cîşting informaţional. În mediul de dezvoltare EXSYS din mulţimea regulilor aplicabile, adică din cele care fac parte din mulţimea conflictuală, se va declanşa efectiv acea regulă care are în partea premisă mai multe condiţii şi în partea acţiune mai multe acţiuni posibile. Logica de mai sus se bazează pe faptul că o astfel de regulă conţine mai multă informaţie, fapt ce va determina atingerea mai repede a obiectivului.

4.2.3.1. Analiza condiţiei

Funcţie de forma condiţiei se va explora contextul pentru a găsi piese de cunoaştere ce corespund clauzelor constituente ale condiţiei. O regulă are o exprimare narativă cum este de exemplu următoarea:

```
IF: domeniul de utilizare al calculatorului este cercetare
    ştiinţifică, sau proiectare asistată
(and) volumul programelor dezvoltate este foarte mare
(and) costul nu reprezintă un factor major

THEN: coprocesor matematic este necesar
(and) frecvenţa ceasului este mare
(and) ploter este necesar
(and) display SVGA.
```

Forma în care o regulă este introdusă în baza de reguli depinde de modul de realizare a interpretorului de reguli. Dacă se folosesc liste înlănţuite cum sînt de exemplu listele LISP condiţia ce determină contextul se exprimă:

```
(Context (domeniul de utilizare al calculatorului este cercetare
         ştiinţifică, sau proiectare asistată) (volumul
         programelor dezvoltate este foarte mare) (costul nu
         reprezintă un factor major)).
```

Exprimarea regulii sub forma unei liste LISP ca în exemplul de mai sus duce la o structură de tipul următor:

REGULA nr. 5

```
(Context  (domeniul de utilizare al calculatorului este cercetare
          ştiinţifică,  sau  proiectare  asistată)  (volumul
          programelor dezvoltate este foarte mare) (costul nu
          reprezintă un factor major))
(Acţiune  (coprocesor matematic este necesar) (frecvenţa ceasului
          este mare) (ploter este necesar) (display SVGA)
```

Într-o reprezentare de acest tip se pot folosi chiar funcţiile LISP pentru corespondenţa cu partea de condiţie cît şi pentru realizarea operaţiilor specifice acţiunii. Acest mod este ineficient din punctul de vedere al memoriei ocupate însă are avantajul simplităţii prin faptul că se utilizează facilităţi ale unui limbaj cunoscut. O astfel de reprezentare a cunoaşterii a fost folosită în sistemul expert MYCIN.

Sistemele expert actuale înglobează mecanisme de evaluare a condiţiei mult mai evoluate fără să se restrîngă la operaţia simplă de corespondenţă între condiţie şi context. Expertul, care este sursa cunoaşterii poate prevedea orice evaluare de funcţie, funcţie care să reprezinte cel mai bine criteriul de acceptare a regulii de producţie, însă reprezentarea nu mai este unică şi modulară. Mecanismele de corespondenţă prezentate pînă în prezent au o dificultate majoră legată de tratarea cunoştinţelor incomplete prin care se modelează enunţurile problemelor, fapt pentru care apare ca necesară introducerea variabilelor. Utilizarea variabilelor face posibil ca în sistemele de producţii să se folosească reguli ce operează nu numai cu piese individuale, ci şi cu concepte. Legarea variabilelor facilitează construcţia de instanţe diferite pentru aceleaşi reguli. În principal prin legarea variabilelor se rezolvă moştenirea proprietăţilor paternurilor din condiţie.

4.2.3.2. Metode de rezolvare a conflictelor

Întrucît nu este cunoscut rezultatul punerii în corespondenţă a condiţiei regulii cu contextul, mai multe reguli sînt selectate în mulţimea candidată şi sistemul va trebui să decidă care dintre acestea vor trebui declanşate. Se poate ajunge la o astfel de situaţie datorită conflictelor între instanţele aceleiaşi reguli cît şi datoria conflictelor între reguli diferite ce candidează la soluţie ca urmare a realizării cu succes a corespondenţei. Rezolvarea acestor conflicte este făcută pe baza informaţiei ce provine din context, din baza de reguli sau surse combinate.

Dacă informaţia provine din context poate fi indicată o anumită importanţă a pieselor de cunoaştere materializată printr-o pondere ce este o dată de intrare în mecanismul de evaluare a priorităţii. Dacă se notează cu p_i ponderea regulii i din baza de reguli următoarele criterii sînt mai des utilizate:

- prioritatea dată de cea mai importantă piesă de cunoaştere din regulă. Pentru alegerea regulii r, $p_r = \max p_i$ în care $1 <= i <= n$;
- prioritatea determinată de media aritmetică a ponderilor de importanţă ce apar în regula

$$p_r = \frac{1}{n} \sum_{i=1}^{n} p_i;$$

- ordinea inversă dispersiei ponderilor de importanţă a pieselor de cunoaştere, ce se calculează prin

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (p_i - p_r)^2$$

în care, p_r este media aritmatică calculată mai sus.

Contextul mai poate furniza informaţii ce nu depind de importanţa pieselor de cunoaştere ci doar de utilitatea şi activarea acestora. Informaţiile de acest tip sînt înregistrate şi actualizate de interpretorul sistemului de producţie şi se referă la:

- vechimea piesei de cunoaştere apreciată după data calendaristică a înscrierii acesteia în context;
- numărul de participări ale piesei de cunoaştere la regulile efectiv declanşate, începînd din momentul înscrierii piesei de cunoaştere;
- numărul de neutilizări, adică declanşări de producţii ce nu folosesc piesa de cunoaştere considerată. Desigur că prin strategiile de rezolvare a conflictelor se apreciază mai mult piesele de cunoaştere mai recente şi cu mai mare număr de utilizări în ultima perioadă.

O sursă de informaţii importantă în ceea ce priveşte rezolvarea conflictelor de prioritate este constituită pe bază de reguli. În acest sens se are în vedere posibilitatea ca la scrierea regulii de producţie să se cunoască informaţii privind modul de determinare a succesorului imediat. Nu este recomandată chemarea explicită întrucît afectează anatomia sistemului de reguli. Nici o altă metodă nu asigură o eficienţă egală cu cea a specificării succesorului. Se încearcă prin diverse metode să se elimine dezavantajul pierderii autonomiei, păstrîndu-se avantajul înlocuirii procesului de căutare printr-o specificaţie directă. O metodă des întîlnită este cea care foloseşte reguli de prestabilire a producţiilor succesoare avînd forma:

(CONFLICT - P
(CONDIŢIE (P))
(ACŢIUNE (SUCCESOR (P_1, P_2,..., P_r)))

unde P este o producţie al cărui succesor se caută, iar (P_1, P_2,..., P_r) este şirul ordonat după prestaţie al candidatelor la producţia succesoare.

O metodă eficientă pentru determinarea succesorului este cea prin care se specifică în regulă obiectivul ce trebuie să-l rezolve regula următoare, eliminînd în acest fel neajunsurile privind chemarea explicită a succesorului.

Interpretorul sistemului de producţie trebuie să-şi memoreze stările determinate prin funcţionarea sa, pentru dirijarea procesului interpretor cît şi pentru acţiunea în caz de eroare.

Stările determinante ale interpretorului sînt similare cu cele folosite în cazul procesoarelor şi anume:

- *stare veche* - se înscrie producţia efectiv declanşată înaintea noului ciclu interpretativ, piesele de cunoaştere implicate în forma veche a contextului ce au suferit modificări;
- *starea curentă* - în care este înscrisă faza din ciclu, producţia ce se evaluează şi piesele de cunoaştere implicate, lista producţiilor implicate în fazele anterioare stării curente, starea acţiunii de rezolvare a conflictelor de prioritate, starea acţiunii în curs de desfăşurare;
- *starea nouă* - este înscrisă producţia de la care se reia ciclul următor.

Informaţiile de stare pot fi folosite pentru rezolvarea conflictelor de prioritate în unul din modurile:

- cercetarea stării vechi spre a vedea dacă noua regulă prioritară este distinctă faţă de ultima regulă declanşată efectiv;
- întrucît strategia precedentă nu elimină în totalitate repetările, se constituie un istoric al declanşărilor efective ce este folosit în procesul de rezolvare a conflictelor de prioritae pentru faza de minimizare a mulţimii conflictuale. Complexitatea informaţiei de stare şi a mecanismului de comutaţie a acesteia, în vedere asigurării unei desfăşurări controlate a ciclului de interpretare a descurajat mulţi autori. Multe din interpretoarele de pînă acum sînt simple fără control de stare care să permită în orice moment rezolvarea condiţiilor excepţionale ce pot apare ca urmare a incompletitudinii contextului, a considerării cererilor urgente, a optimizării folosirii resurselor, a ridicării gradului de paralelism la efectuarea proceselor.

Rezolvarea conflictelor de prioritate se poate baza şi pe faptul că structura regulilor de producţie selectate este purtătoare de informaţie utilă pentru acest scop. O regulă P_P este un caz particular al unei reguli P_G dacă:

- P_P şi P_G sînt identice;
- toate clauzele condiţionale ale regulii generale P_G se regăsesc printre clauzele condiţionale ale regulii P_P;
- pentru fiecare clauză condiţională din P_G ce specifică piese de cunoaştere în context, există în P_P o clauză care conţine un subset de piese de cunoaştere, printre care şi aceea specificată în clauza din P_G.

În acest caz se preferă P_P pentru a satisface în întregime restricţiile specificate de regula P_G.

4.2.3.3. Transmiterea acţiunii

Este caracteristică pentru sistemele de producţie comunicarea prin intermediul contextului între reguli. Metoda este folosită în programarea clasică, avînd ca principiu prevederea unui depozit de informaţie la dispoziţia mai multor taskuri. Această zonă se mai numeşte şi zonă de comunicaţie sau cutie poştală. Metoda întîlnită în literatura numită şi "comunicaţie prin date", abordează programul nu ca pe o secvenţă prestabilită de operaţii, ci realizează înlănţuirea operaţiilor ca urmare a interpretării conţinutului cutiei poştale după reguli ce permit stabilirea relaţiei următoare. Aceste reguli pot substitui astfel mecanismele de control ale calculatorului, evident nu la nivelul instrucţiunilor elementare.

În consecinţă principala funcţie a părţii de acţiune a regulii este cea de realizare a funcţiei de comunicare, adică de înscriere a informaţiei pe care regula o produce în context. Se pot aminti principalele operaţii ce se execută asupra contextului cum sînt:

- ştergerea unei piese de cunoaştere din context, cu acţiune atît asupra piesei şi de cunoaştere cît şi asupra informaţiei auxiliare menţinută de sistem;
- înscrierea unei piese de cunoaştere în context cît şi declanşarea unor proceduri de evaluare a priorităţilor pentru plasarea noii piese de cunoaştere în mulţimea ordonată a contextului;
- modificarea unei piese de cunoaştere este o operaţie ce porneşte de la forma veche a piesei de cunoaştere (informaţie ce se utilizează la recunoaşterea acesteia în context) ca prim operand, al doilea operand fiind dat de o nouă formă a piesei de cunoaştere înscrisă în context pe aceeaşi poziţie, fără a se schimba informaţia asupra priorităţii piesei de cunoaştere.

O serie de operaţii auxiliare sînt posibile, cum ar fi:

- activarea sau dezactivarea unor reguli de producţie, acţionînd asupra potenţialului de cunoaştere procedurală;
- alterarea conţinutului bazei de reguli, prin ştergerea, înscrierea sau modificarea acestora;
- iniţierea executării unor programe ce au ca efect:
- execuţia unor secvenţe de operaţii ce nu afectează contextul;
- secvenţe de operaţii ce modifică strategia de control cum sînt apelul explicit al regulii, testări ce folosesc la diverse procese decizionale diferite de cele implementate prin strategia decizională a interpretorului;
- efectuarea unor calcule prin asignarea de valori unor simboluri.

Datorită caracteristicilor structurale ale acestor operaţii este uneori necesar să se dispună şi de operatori cu caracter constructiv cu ajutorul cărora pot fi construite noi piese de cunoaştere cu structuri adecvate; construcţii de liste cu obţinerea de contexte locale.

4.2.3.4. Organizarea sistemelor de producţii

Metoda de reprezentare a cunoaşterii cu ajutorul sistemelor de producţii a avut un mare succes datorită simplităţii regulilor, a modularităţii acestora, formatului liber al pieselor de cunoaştere, aspectului natural al transpunerii în reguli de acest tip a cunoaşterii. Simplitatea principială a metodei se pierde atunci cînd se elimină deficienţele, determinînd o mare complexitate a sistemului.

Comunicaţia regulilor prin intermediul contextului face ca modificarea conţinutului unei reguli să aibă influenţă doar asupra condiţiei de aplicabilitate şi a concluziei. Sistemele de producţii sînt caracterizate de o autonomie ridicată ce face ca regulile să fie tratate ca piese de cunoaştere independente. Un sistem de producţii instruit prin reguli şi fapte poate să rezolve un anumit tip de probleme într-un interval de tip Dt_0. Dacă sistemul este instruit să rezolve şi alte probleme, aceasta determină ca problema iniţială să se rezolve într-un timp $Dt_1 > \Delta t_0$. Rezultă că în urma acumulării cunoaşterii sistemul devine din ce în ce mai neperformant, şi deci

$$Dt_n > Dt_{n-1} > \ldots > Dt_1 > Dt_0.$$

Modul de lucru cu sistemele de producţii este adecvat aplicaţiilor în domeniile în care cunoaşterea este difuză, secvenţe de acţiuni nu pot fi determinate. Sînt puţin avantajate pentru reprezentarea prin această metodă situaţiile deterministe exprimabile prin algoritmi sau prin fluxuri predominante de control.

4.2.3.5. Măsuri pentru creşterea performanţelor

Utilizarea sistemelor de producţii face necesară o temeinică analiză în scopul prevenirii şi eliminării cauzelor ce conduc la deprecierea performanţelor sistemului. Aceste măsuri au în vedere:

- organizarea contextului pentru a se permite o explorare parţială;
- organizarea bazei de reguli avînd ca scop eliminarea în procesul inferenţei a regulilor neadecvate problemei de rezolvat;
- înglobarea de funcţii auxiliare ce facilitează accesul, selecţia, execuţia şi comunicarea.

Organizarea contextului priveşte obţinerea unor grupuri de piese de cunoaştere adecvate rezolvării unor tipuri predefinite de cunoaştere, adecvate rezolvării unor tipuri predefinite de probleme, context ce va fi denumit *context local*. Învăţarea din experienţă este o activitate teoretic posibilă dar practic

foarte dificilă. Informaţia referitoare la tipurile de probleme în care o piesă de cunoaştere este implicată nu este suficientă pentru preselecţie, însă folosită pentru preordonarea contextului face ca şansele de reuşită să crească simţitor.

Specializarea funcţională a regulilor priveşte grupări ce se realizează pe baza similitudinii obiectivelor urmărite la aplicarea regulilor de producţie individuale, adăugarea de noi contexte prin care sistemul reţine actuala informaţie necesară eficientizării procesului de căutare face să se restrîngă căutarea numai la acele entităţi ce pot candida la încheierea cu succes a fazei. Aceste contexte sînt numite *contexte de referinţe*. În faza de testare a condiţiilor este util să se cunoască care sînt regulile dintr-o bază de reguli dată, ce se încearcă într-un context dat. Se constituie astfel un context de referinţe ce informează asupra producţiilor în ale căror condiţii se află o anumită piesă de cunoaştere din contextul global. Fiecare înregistrare în contextul de referinţe privind apartenenţa la partea de condiţii va fi legată de o piesă distinctă de cunoaştere din contextul global şi va avea structura:

[<piesa de cunoaştere> <lista de reguli>].

Într-un context local specific unei probleme şi la o specializare corespunzătoare în baza de reguli pot fi eliminate căutările inutile. Astfel, prin extragerea din contextul de referinţe a acelora ce corespund pieselor de cunoaştere din contextul local, reunirea listelor de reguli asociate, obţinerea listei regulilor active pentru primul pas al ciclului prin intersecţia între lista reunită din pasul precedent şi acţiunea specializată pe problema din baza de reguli, eficienţa procesului va creşte.

Un alt context de referinţe poate conţine pentru fiecare regulă lista de referinţă la piesele de cunoaştere existente în partea de condiţie cu structura de mai jos

[<regula> <listă piese cunoaştere>].

Utilizarea acestui context presupune intersecţia referinţelor din context ce corespund regulilor active cu contextul local, după care regulile ce menţin conţinutul listei de piese de cunoaştere asociate vor fi selectate pentru faza de corespondenţă a ciclului. Sistemele ce utilizează contexte de referinţă obţin performanţe mult mai bune comparativ cu sisteme tradiţionale ajungînd de pînă la şase ori mai mari. Componentele interpretorului de reguli ce prelucrează contextele de referinţe sînt denumite *filtre*.

Metoda împărţirii bazei de reguli în grupuri independente de reguli de producţie specializate, în efectuarea unor acţiuni prestabilite, în soluţionarea unor probleme specifice determină noţiunea de *sursă de cunoaştere*. O sursă de cunoaştere acţionează independent de celelalte asupra pieselor de cunoaştere şi produce date noi numite ipoteze ce sînt fie confirmări, fie obiecte noi. Ipotezele se depun într-o zonă de memorie comună numită şi tablă (blackboard). O tablă este împărţită în planuri ce reprezintă zone special dedicate unor categorii de piese de cunoaştere. În cadrul unui plan se pot partiţiona nivele ce corespund surselor de cunoaştere distincte.

Construcţia bazei de cunoştinţe astfel încît procesul inferenţial să fie cît mai scurt poate fi realizată după un algoritm ce porneşte de la ideea algoritmului ID3 propus de Rose Quinlan. Se dă mai jos exemplul unei baze de cunoştinţe specifică unui sistem expert cu aplicaţii în autodiagnoză, cu invatare din exemple. Se pune problema organizării în forma cea mai adecvată a cunoştinţelor. Se generează reguli de clasificare sub formă de arbori de decizie pe baza unui set de exemple. Notînd cu t_1, t_2, t_3 teste care după execuţie întorc valori în domenii cunoscute cum sînt (t_{1a}, t_{1b}, t_{1c}) (t_{2a}, t_{2b}, t_{2c}) (t_{3a}, t_{2b}) şi d_1, d_2 clase de defecte posibile obţinem exemple de defectare ca:

$$t_{1a}, t_{2a}, t_{3b} : d_1$$
$$t_{1a}, t_{2c}, t_{3a} : d_1$$
$$t_{1c}, t_{2a}, t_{3a} : d_2$$
$$\ldots\ldots\ldots\ldots\ldots$$

Exemplele date mai sus vor fi citite astfel: dacă t_1 întoarce rezultatul t_{1a}, t_2 întoarce rezultatul t_{2a}, t_3 întoarce rezultatul t_{3b} atunci se constată defectul aparţinînd clasei defectelor de tip d_1. Pot fi obţinuţi astfel mai mulţi arbori corecţi din punct de vedere logic, dar avînd lungimi diferite în funcţie de nodul iniţial. Arborii diferă prin ordinea nodurilor de decizie ce reprezintă formal ordinea de execuţie a testelor. Pentru micşorarea timpului de atingere a scopului, înălţimea arboreului trebuie să fie cît mai mică. Considerînd următoarele atribute cu valorile lor:

a_1 cu valorile $a_{11}, a_{12},\ldots, a_{1p}$
a_2 cu valorile $a_{21}, a_{22},\ldots, a_{2k}$
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$
a_n cu valorile $a_{n1}, a_{n2},\ldots, a_{nj}$

şi obiectele, (aici defecte) $d_1, d_2,\ldots, d_q$ ce sînt caracterizate de atributele $a_1, a_2,\ldots, a_n$, cantitatea de informaţie din setul de exemple este dată de relaţia

$$C = -P_{d1} * \log (P_{d1}) - P_{d2} * \log (P_{d2}) - \ldots - P_{dn} * \log (P_{dn})$$

în care, P_{di} reprezintă probabilitatea de apariţie a lui d_i în setul valorilor $d_1, d_2,\ldots, d_n$ obţinîndu-se P_{di} = numărul de apariţii ale lui d_i/n.

Atributul a_i împarte prin valorile sale $a_{i1}, a_{i2},\ldots, a_{ik}$ setul de exemple în clase. Dacă există:

$n1$ obiecte cu valoarea atributului $a_i = a_{i1}$
$n2$ obiecte cu valoarea atributului $a_i = a_{i2}$
$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$

şi notînd cu n numărul iniţial de obiecte, se obţine cîştigul de informaţie dat de alegerea atributului a_i pentru formarea nodului de decizie ca fiind $C - C(a_i)$, în care

$$C(a_i) = (n1/n) * C1 + (n2/n) * C2 + \ldots$$

unde Ci reprezintă "cantitatea de informaţie" în fiecare din clasele de exemple obţinute prin partitionarea după un atribut. Atunci se va alege ca atribut iniţial de decizie acel atribut pentru care valoarea $C - C(a_i)$ este maximă. Pentru colecţia de teste t_1, t_2, t_3 dacă atributul t_2 are cîştig de informaţie faţă de t_1 atunci t_2 îl va precede pe t_1.

Procedura de construcţie arbore este ilustrată in pseudocod mai jos.

```
   //ex = set exemple iniţial
   //arb = arbore de decizie obţinut
atribuie set (ex, null)
cît timp setul nu este vid repetă
   atribuie m-alege element (set) //m-colecţie de exemple cu arborele
   aferent
   dacă contradicţie (m) atunci //verifică dacă mulţimea aleasă
   conţine //obiecte de tip diferit
   atribuie a--alege-atribut (m) //se alege atributul cu cel mai mare
   "cîştig"
   atribuie ma formează mulţimi (a, m) //se obţin clasele în care se
   împarte //setul de exemple funcţie de valoarea lui a
dacă nod = null atunci
   atribuie nod, formează nod (a) //formează rădăcina arborelui
altfel completează nod cu atribut (nod, a)
pentru succesiunea aparţinînd ma repeta
   atribuie  nod p <- obţine nod
         set <- set + (succesorii, nod p)
   completare nod cu fii (nod, nod p)
altfel atribuie nod selectează nod (m)
   completează nod cu mesaj de sfîrşit parcurgere arbore şi elementul
   la care s-a ajuns
sfîrşit.
```

Se consideră la acest moment un caz ipotetic de defectare pentru care se exemplifică organizarea după atributul t_1. Acest atribut împarte setul iniţial de exemple în două clase ce sînt caracterizate de rezultatul întors de t_1, adică t_{1a} şi t_{1b}. Clasa caracterizată de t_{1a} este

$t_{1a}, t_{2c}, t_{3b} : d_2$
$t_{1a}, t_{2b}, t_{3a} : d_1$
$t_{1a}, t_{2a}, t_{3a} : d_2$
$t_{1a}, t_{2c}, t_{3a} : d_1$
$t_{1a}, t_{2a}, t_{3b} : d_2$

şi clasa caracterizată de t_{1b}

$t_{1b}, t_{2c}, t_{3a} : d_1$
$t_{1b}, t_{2a}, t_{3a} : d_2$
$t_{1b}, t_{2c}, t_{3b} : d_2$.

Se va obţine pentru cîştigul informaţional calculat după relaţiile de mai sus:

$C = -3/8 * \log_2 3/8 - 5/8 * \log_2 5/8 = 0.954$
$C1 = -2/5 * \log_2 2/5 - 3/5 * \log_2 3/5 = 0.971$
$C2 = -1/3 * \log_2 1/3 - 2/3 * \log_2 2/3 = 0.918$
$C(t_1) = 5/8 * 0.971 + 3/8 * 0.918 = 0.951$
$C - C(t_1) = 0.003.$

Dacă clasificarea se face după valorile atributului t_2 se vor obţine rezultatele:

$t_{2a}, t_{1a}, t_{3a} : d_2$
$t_{2a}, t_{1a}, t_{3b} : d_2$
$t_{2a}, t_{1b}, t_{3a} : d_2$

clasa caracterizată de t_{2b}

$t_{2b}, t_{1a}, t_{3a} : d_1$

şi clasa caracterizată de t_{2c}

$t_{2c}, t_{1a}, t_{3b} : d_2$

$t_{2c}, t_{1a}, t_{3a} : d_1$

$t_{2c}, t_{1b}, t_{3a} : d_1$

$t_{2c}, t_{1b}, t_{3b} : d_2$.

Va rezulta:

$C1 = -3/3 * \log_2 3/3 = 0.000$

$C2 = -1/1 * \log_2 1/1 = 0.000$

$C3 = -2/4 * \log_2 2/4 - 2/4 * \log_2 2/4 = 0.500$

$C(t_2) = 3/8 * 0.000 + 1/8 * 0.000 + 4/8 * 0.500 = 0.250$

$C - C(t_2) = 0.704$

Este evident că atributul t_2 are un cîştig net de informaţie faţă de atributul t_1 fapt pentru care regulile ce se grupează după acest atribut vor trebui să le preceadă pe cele grupate după atributul t_1.

Alte cunoştinţe de o importanţă deosebită conţinute în baza de cunoştinţe se referă la ordinea de execuţie a testelor. Întrucît nu se dispune în general de un analizator de semnătură, ce ar fi foarte util pentru testarea magistralelor şi a bufferelor, o ordine de execuţie a testelor bine gîndită va ajuta depistarea defectelor. Spre exemplu dacă este posibil accesul la date pe cele două părţi ale magistrelei (lo sau hi) se vor testa ambele independent, apoi global coroborînd concluziile. De asemenea, în sistemele cu procesoare multiple, procesorul master va impune ordinea testelor plachetelor subordonate.

4.2.3.6. Inconsistenţa sistemelor de producţii

Cele mai multe aplicaţii sînt realizate cu sisteme bazate pe reguli de producţie din cauză că există o gamă destul de largă de programe cum sînt ESE (IBM), Personal Consultant Plus (Texas Instruments) pentru construcţia sistemelor expert bazate pe acest mod de reprezentare. Ele sînt în general succesoare ale sistemului expert EMYCIN şi au aceeaşi structură. O problemă importantă este cea a obţinerii sistemelor expert cu o mare cantitate de cunoaştere, situaţie în care este greu să se menţină consistenţa bazei de cunoştinţe.

Modul în care se trage o concluzie într-un sistem de producţii se mai numeşte şi *derivaţie*. Pentru un sistem de producţii $R = (D, P)$ şi $R' = (D', P)$

se defineşte $(D, P) \overset{1}{\underset{PS}{\vdash}} (D', P)$ dacă şi numai dacă există regula IF c THEN t regulă ce aparţine lui P cu c component al lui D, de unde rezultă că $D' = D \cup \{t\}$. Se va nota $\overset{1}{\underset{PS}{\vdash}}$ închiderea tranzitivă şi reflexivă.

Definiţie: O bază de date D este inconsistentă dacă şi numai dacă există termeni $x = a$ şi $x \neq a$ ca elemente în D.

Fie ca exemplu o bază de date D a unui sistem de producţii cu D = {cutie = mare rezistenţă, şurub = şurub ascuns, şurub ≠ şurub ascuns}

Cum şurub = şurub ascuns şi şurub ≠ şurub ascuns sînt două elemente în D rezultă că D este inconsistenţa.

Definiţie: Un sistem de producţii $R = (D, P)$ este inconsistent dacă şi numai dacă există o bază de date D' şi $(D, P) \underset{PS}{\vdash} (D', P)$, iar D' este inconsistent. Cu alte cuvinte pentru o bază de date consistentă D în sistemul de producţii (D, P) se poate obţine o bază inconsistentă în urma aplicării derivatei.

Se consideră mai jos inconsistenţa într-un mod foarte limitat legat de posibilitatea de a deduce contradicţii în sistemele de producţie. Ca exemplu se porneşte de la următorul sistem de producţii $R = (D, P)$, în care

D = {Construcţie = rezistenţă, cutie = aspectoasă, conectare = puternică}

P = {P_1 : IF construcţie = rezistenţă cutia = aspectuoasă THEN confecţie = aluminiu

P_2 : IF cutia = aspectuoasă THEN şurub = şurub cap îngropat

P_3 : IF conectare = puternică THEN şurub ≠ şurub cu cap îngropat}.

Desigur că chiar dacă D este consistent se va obţine

D = {Construcţie rezistenţă, cutie = aspectuoasă, conectare = puternică, confecţie = aluminiu, şurub = şurub cu cap îngropat, şurub ≠ şurub cu cap îngropat}.

În sisteme de producţii pot fi avute în vedere şi alte restricţii pornind de la faptul că un parametru poate avea o singură valoare sau mai multe. Restricţia prin care un parametru are valoare unică poate fi exprimată ca regulă. Fie $R = (D, P)$ un sistem de producţii cu $(R) = \{x_1, ..., x_m\}$ şi $(P) = \{a_1, ..., a_n\}$ şi restricţia ca x_k are o singură valoare în P. Se poate asimila sistemul de producţii cu sistemul $R' = (D, P')$, unde

$$R' = R \cup \{\text{IF } x_k = a_i \text{ THEN } x_k \neq a_j \mid i, j = \overline{1, n} \text{ şi } i \neq j\}$$

Se obţine astfel că R este inconsistent dacă şi numai dacă $R' = (D, P')$ este inconsistent. În acest mod se poate considera că valorile multiple sînt posibile şi în plus se dă un set de reguli pentru parametrii ce au o singură valoare.

Consistenţa sistemelor de producţie fără negare poate fi determinată utilizînd algoritmul de mai jos. Fie $R = (D, P)$ un sistem de producţii fără negare. Pornind de la baza de date D se generează toţi termenii ce pot fi cuprinşi în R.

Algoritm:

$D'' = D$

$D' = \text{null}$

while $(D'! = D'')$

$D' = D''$

$P'' = \{(\text{IF } c \text{ THEN } t) \in P \mid c \text{ este adevărat pentru } D''\}$

$D'' = D'' \cup \{t \mid (\text{IF } c \text{ THEN } t) \in P''\}$

if $((\exists\, x = a \;\&\; x! = a) \in D'')$

then (D, P) este inconsistent

else

(D, P) este consistent.

Finitudea algoritmului se bazează pe faptul că numărul regulilor sistemului de producţii este finit.

4.2.4. Reţele semantice

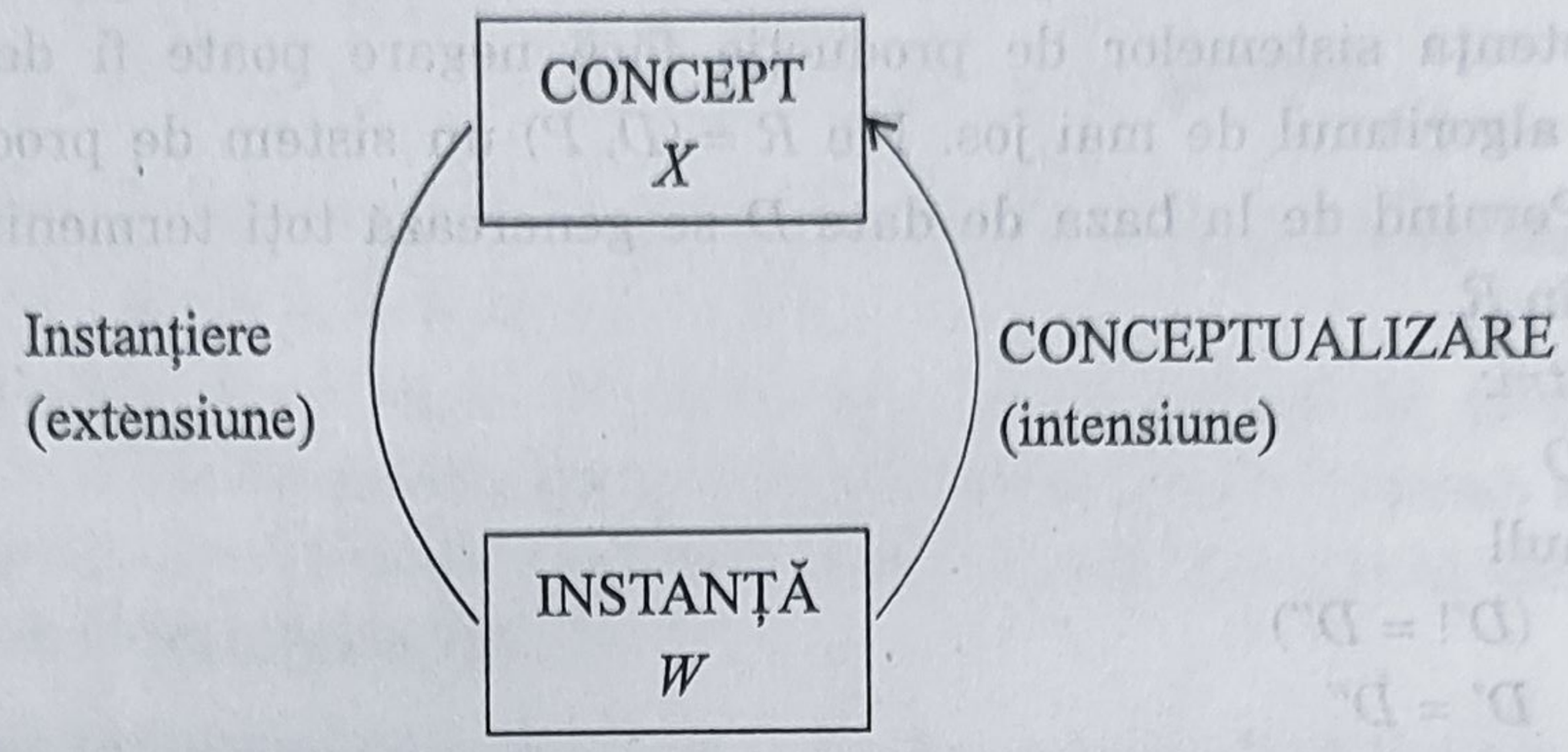
Acest mod de reprezentare a apărut ca o consecinţă a modului de surprindere a structurilor relaţionale de mare complexitate. Aspectul de graf al reprezentării constituie doar o variantă de evoluţie pentru caracteristicile puse în relief prin reprezentarea cunoaşterii cu predicate de ordinul întîi.

Elementele primitive constituente ale unei reţele semantice sînt:

- *noduri* - abstractizări structurale pentru concepte, evenimente, stări, alte obiecte suport;
- *arce* (legături) - abstractizări ale relaţiilor propriu-zise.

Primitivele de acest fel sînt utilizate în limbajele de nivel înalt ce se folosesc pentru reprezentarea structurilor de date - de spaţii de memorie numite şi celule, legate între ele prin pointeri (ex. LISP). Simbolurile şi valorile asociate structurilor de date permit exprimarea unor semnificaţii şi apropierea de cerinţele sistemelor de reprezentarea cunoaşterii.

▼ ***Fig. 4.8. Relaţia concept instantă***



Modelul de reprezentare prin reţele semantice a fost propus în 1966 de Ross Quillian şi s-a numit "reţea semantică". Denumirea se explică prin faptul că reproduce din punct de vedere conceptual structura semantică a dicţionarelor explicative cele două componente fiind:

- noduri - concepte reprezentate prin cuvinte;
- arce - relaţii cu alte concepte.

Denumirea de reţea semantică nu corespunde calităţilor metodei, motiv pentru care se propune folosirea termenului de reţea asociativă (apropierea de memoria umană). Se menţine denumirea de reţea semantică ce cuprinde o varietate de metode ce folosesc structura de reţea pentru construirea bazelor de cunoştinţe destinate programelor ce realizează o serie de efecte caracteristice procesului de înţelegere. Se poate utiliza noţiunea de "limbajul unor reţele" ce exprimă concepte, evenimente, stări, episoade şi relaţii între acestea. Pentru limbajul unei reţele se utilizează o serie de simboluri care sînt înţelese de interpretorul reţelei, simboluri ce formează primitivele semantice. Combinarea primitivelor conform regulilor limbajului determină expresii.

Descrierea este efectuată prin *obiecte formale intensionale denumite şi concepte* cît şi prin obiecte *extensionale denumite instante*. Relaţia între obiecte şi instante este denumită şi *instanţiere sau denotare* şi are caracter extensional pe cînd relaţia între instante şi concepte denumită şi *conceptualizare are un caracter intensional* (fig. 4.8).

Aceasta se poate reprezenta cu ajutorul limbajului calculului cu predicate de ordinul întîi:

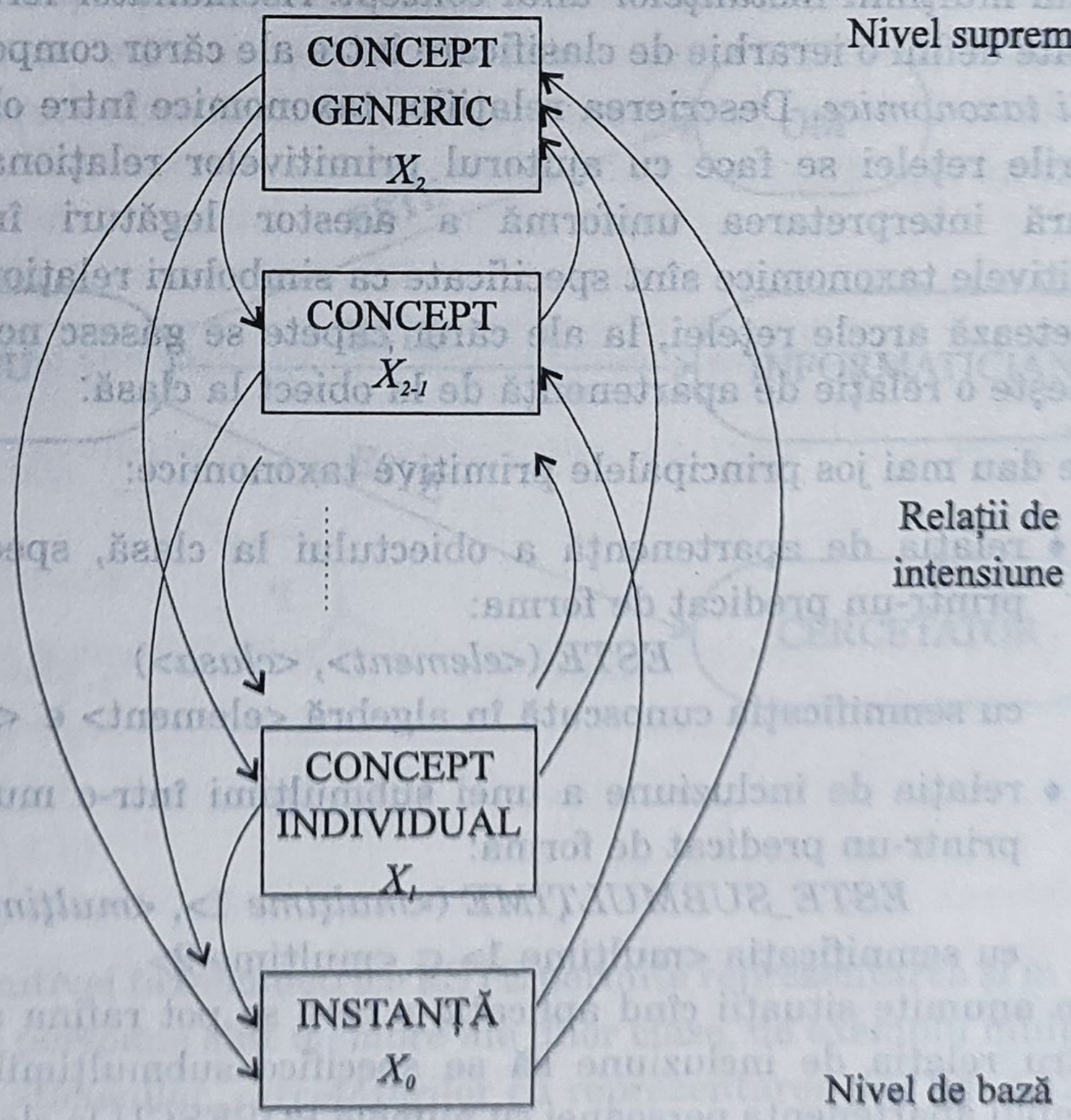
INTENSIUNE (CONCEPT (x), INSTANTA (w)) indicînd faptul că un concept este o intensiune a instantei sale.

EXTENSIUNE (INSTANTA (w), CONCEPT (x)) se specifică faptul că o instantă este o extensiune a conceptului, în care:

CONCEPT (x) este funcţional ce ia valori în mulţimea conceptelor.
INSTANTA (w) este un funcţional ce ia valori în mulţimea instanţelor.

O instantă la rîndul său poate reprezenta un obiect din lumea reală sau un alt concept. Atunci cînd instanta este un concept poate avea la rîndul său instanţe. Se obţine astfel o ierarhie conceptuală din mai multe nivele. Nivelul de bază al ierarhiei conceptuale este denumit şi nivel *infim* şi este totdeauna o instantă ce reprezintă un obiect al lumii reale. Toate nivelele unei ierarhii conceptuale sînt concepte generice cu excepţia penultimului nivel care este un concept individual, concept ale cărui instante sînt indivizi (fig. 4.9).

▼ ***Fig. 4.9. Relaţii într-o ierarhie conceptuală***



Conceptul fiind de fapt un obiect cu structură este alcătuit din entităţi formale numite componente împreună cu relaţiile dintre acestea. Componentele fiind dependente de concept, natura lor poate fi aceeaşi cu a conceptului ca părţi ale unui întreg, sau diferită de a conceptului ca atribute ale unor obiecte. Orice instanţă a unui concept are o structură relaţională ca cea specificată pentru conceptul respectiv.

Natura relaţiilor poate fi de tipul:

- *relaţii epistemice* - permit descrierea semnificaţiei unui concept cu ajutorul unor concepte mai puţin generale;
- *relaţii compoziţionale* - între concepte şi componentele lor;
- *relaţii de apartenenţă* - a conceptelor sau instanţelor la clase sau mulţimi;
- *relaţii cauzale* - între concepte de tip obiect şi concepte de tip acţiune;
- *relaţii modale* - între componente şi modalităţile de ataşare a acestora la concepte.

De multe ori este greu a se formula un concept dar se pot descrie clase ce aparţin mulţimii instanţelor unui concept. Asemănător ierarhiei conceptuale se poate defini o ierarhie de clasificare între ale căror componente se stabilesc *relaţii taxonomice*. Descrierea relaţiilor taxonomice între obiectele plasate în nodurile reţelei se face cu ajutorul primitivelor relaţionale taxonomice ce asigură interpretarea uniformă a acestor legături în toată reţeaua. Primitivele taxonomice sînt specificate ca simboluri relaţionale (predicate) ce etichetează arcele reţelei, la ale căror capete se găsesc noduri între care se defineşte o relaţie de apartenenţă de la obiect la clasă.

Se dau mai jos principalele primitive taxonomice:

- relaţia de apartenenţă a obiectului la clasă, specificată de regulă printr-un predicat de forma:

  ESTE (<*element*>, <*clasa*>)

  cu semnificaţia cunoscută în algebră <element> ∈ <clasa>;

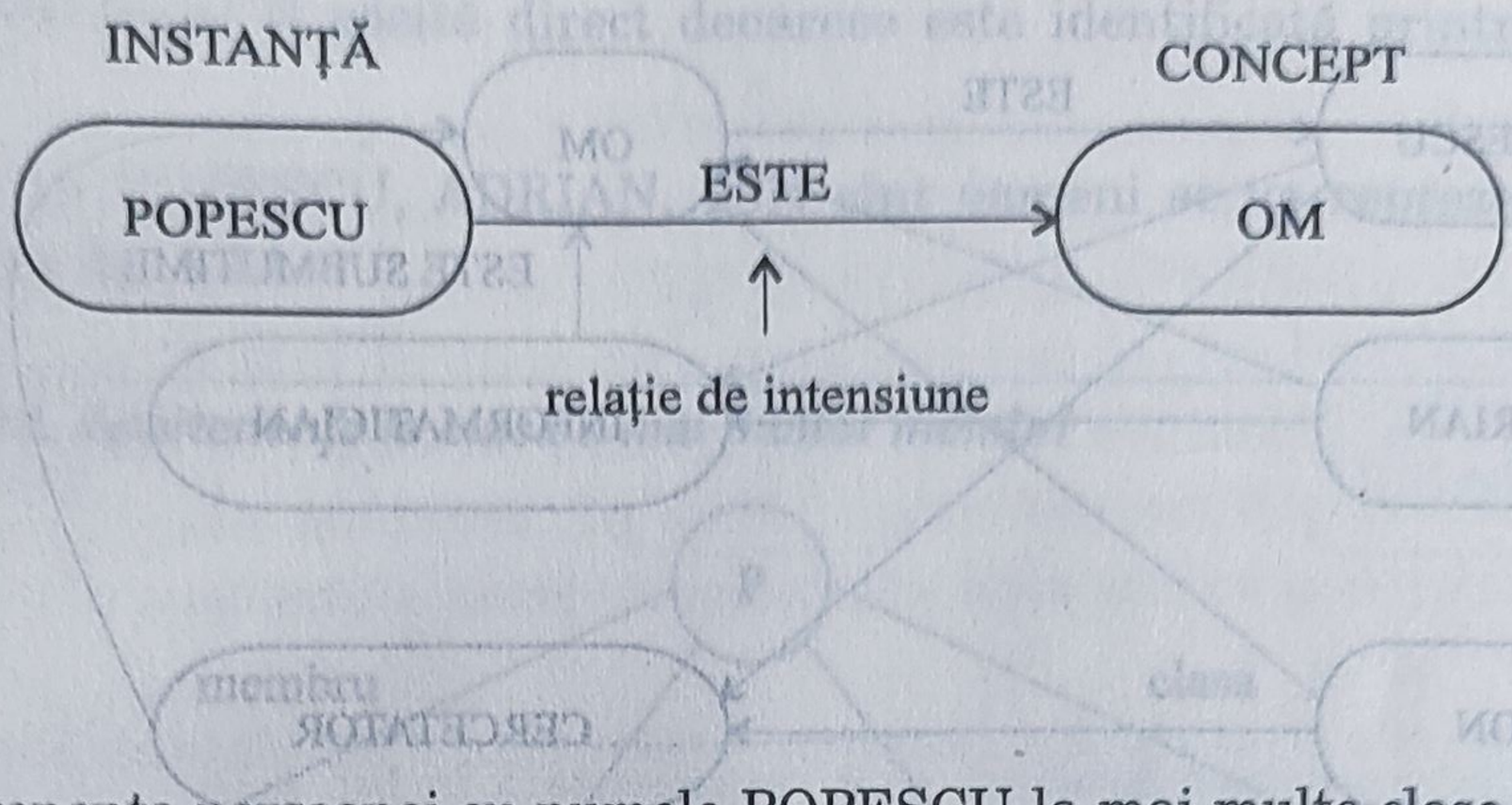
- relaţia de incluziune a unei submulţimi într-o mulţime se specifică printr-un predicat de forma:

  ESTE_SUBMULŢIME (<*mulţime* 1>, <*mulţime* 2>)

  cu semnificaţia <mulţime 1> ⊂ <mulţime 2>.

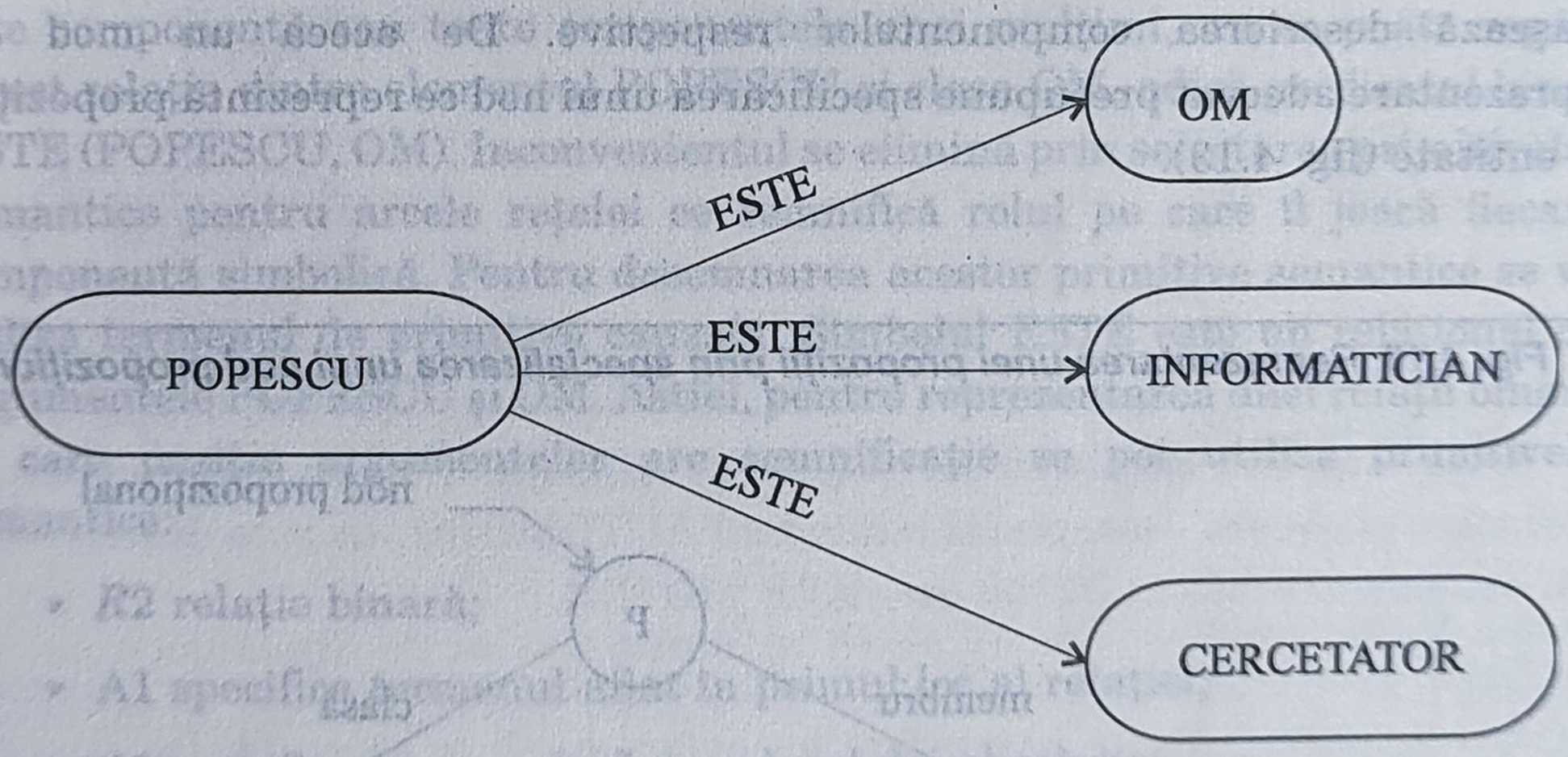
În anumite situaţii cînd aplicaţia o cere se pot rafina aceste relaţii încît pentru relaţia de incluziune să se specifice submulţimile disjuncte. Spre exemplu apartenenţa persoanei cu numele POPESCU la clasa oamenilor ce se exprimă cu ajutorul propoziţiei POPESCU ESTE OM se va transpune într-o reţea semantică cu forma reprezentată în fig. 4.10.

▼ ***Fig. 4.10. Relaţia de apartenenţă***



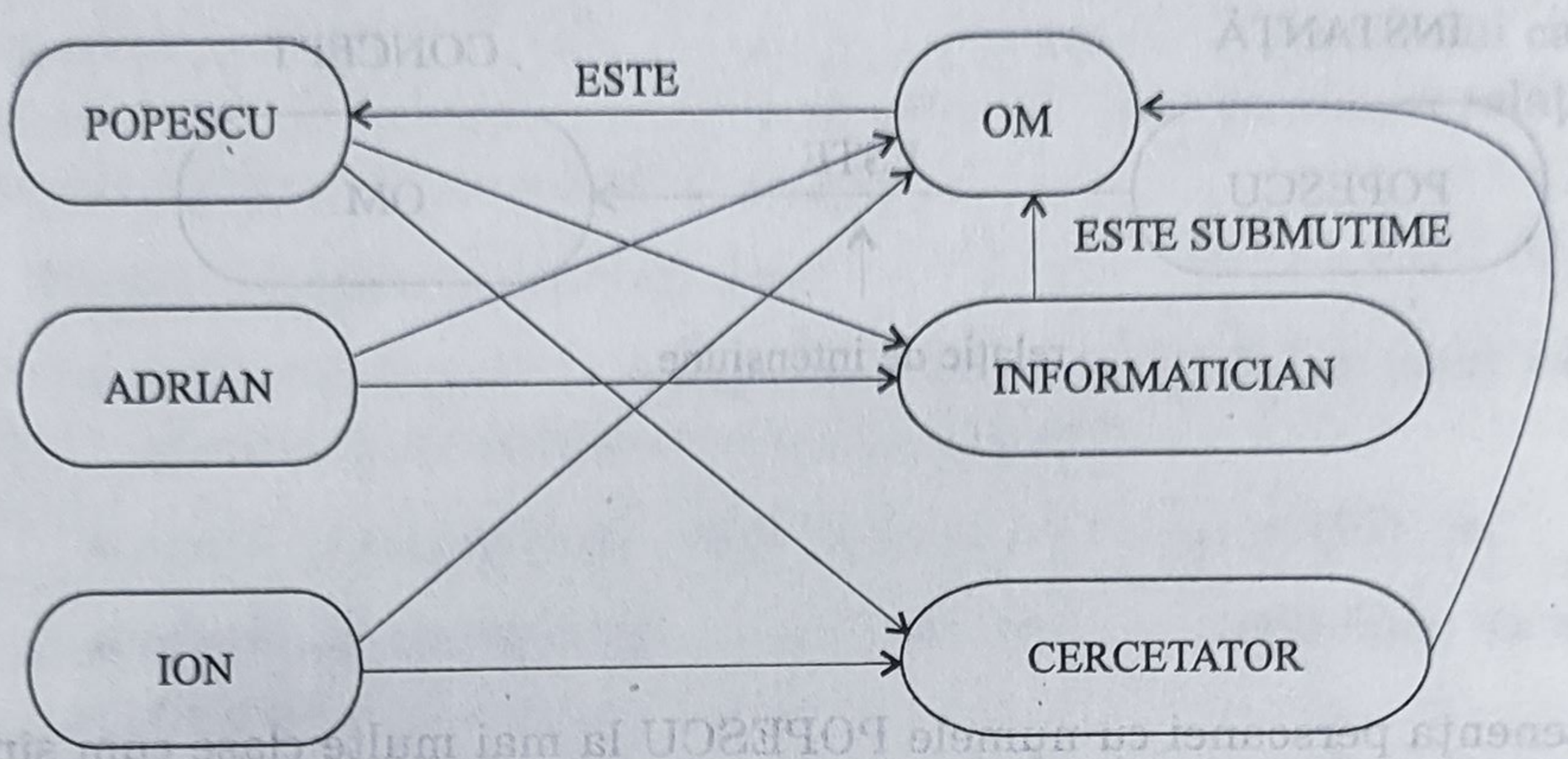
Apartenenţa persoanei cu numele POPESCU la mai multe clase cum sînt mulţimea informaticienilor, a cercetătorilor şi a oamenilor este reprezentată în fig. 4.11.

▼ ***Fig. 4.11. Utilizarea primitivei taxonomice este.***



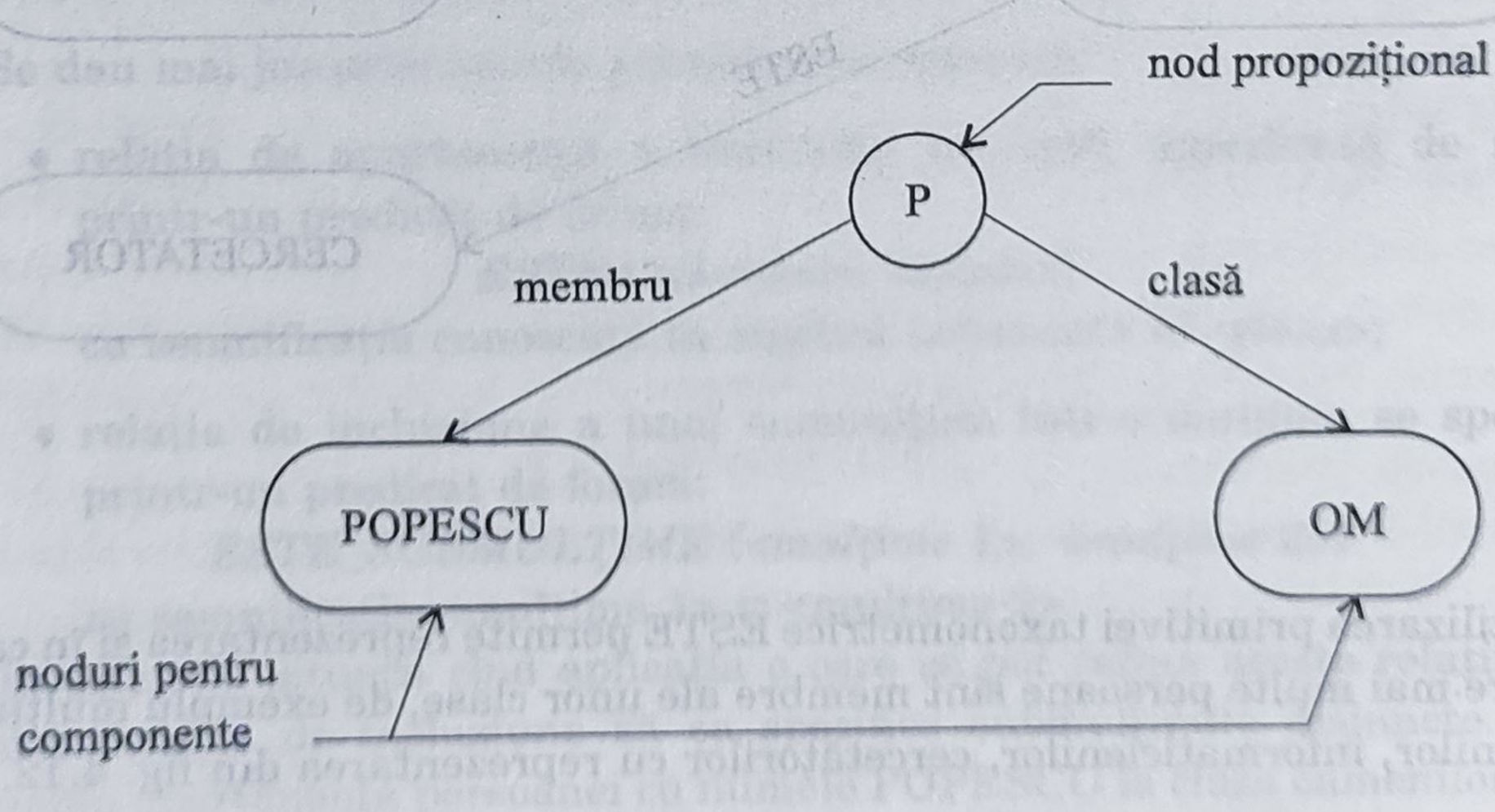
Utilizarea primitivei taxonometrice ESTE permite reprezentarea şi în cazul în care mai multe persoane sînt membre ale unor clase, de exemplu mulţimea oamenilor, informaticienilor, cercetătorilor cu reprezentarea din fig. 4.12.

▼ ***Fig. 4.12. Utilizarea primitivei taxonomice este în reprezentarea instantelor şi submulţimilor***



Cu toate că reprezentarea este sugestivă este însă dificil de regăsit direct propoziţia POPESCU ESTE OM sau altă propoziţie corespunzătoare relaţiilor specificate, întrucît nu au fost reprezentate noduri propoziţionale la care se ataşează descrierea componentelor respective. De aceea un mod de reprezentare adecvat presupune specificarea unui nod ce reprezintă propoziţia ca entitate (fig. 4.13).

▼ ***Fig. 4.13. Reprezentarea unei propoziţii prin specializarea unui nod propoziţional***



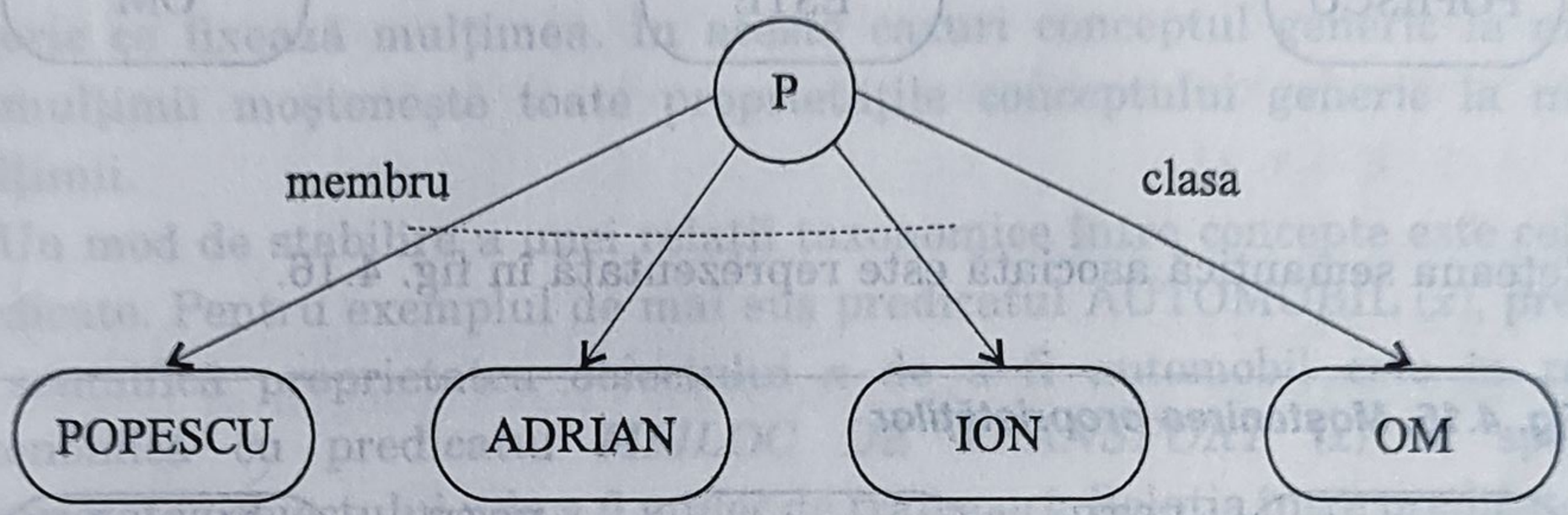
Astfel, propoziţia se poate interpreta în forma predicatelor de ordinul întîi prin formula:

P = MEMBRU (POPESCU) CLASA (OM)

aserţiune ce poate fi găsită direct deoarece este identificată printr-un unic simbol *P*.

Faptul că POPESCU, ADRIAN, ION sînt oameni se va reprezenta prin reţeaua din fig. 4.14.

▼ ***Fig. 4.14. Apartenenţa la clasă a mai multor membri***



Chiar dacă în acest mod se pot găsi toate submulţimile la care un element este componentă sau toate componentele unei mulţimi, nu se poate regăsi direct relaţia dintre elementul POPESCU şi clasa OM, adică predicatul binar ESTE (POPESCU, OM). Inconvenientul se elimină prin selectarea primitivelor semantice pentru arcele reţelei ce semnifică rolul pe care îl joacă fiecare componentă simbolică. Pentru desemnarea acestor primitive semantice se va utiliza termenul de primitivă cauzală. Simbolul ESTE este un relaţional cu argumentele POPESCU şi OM. Astfel, pentru reprezentarea unei relaţii binare în care poziţia argumentelor are semnificaţie se pot utiliza primitivele semantice:

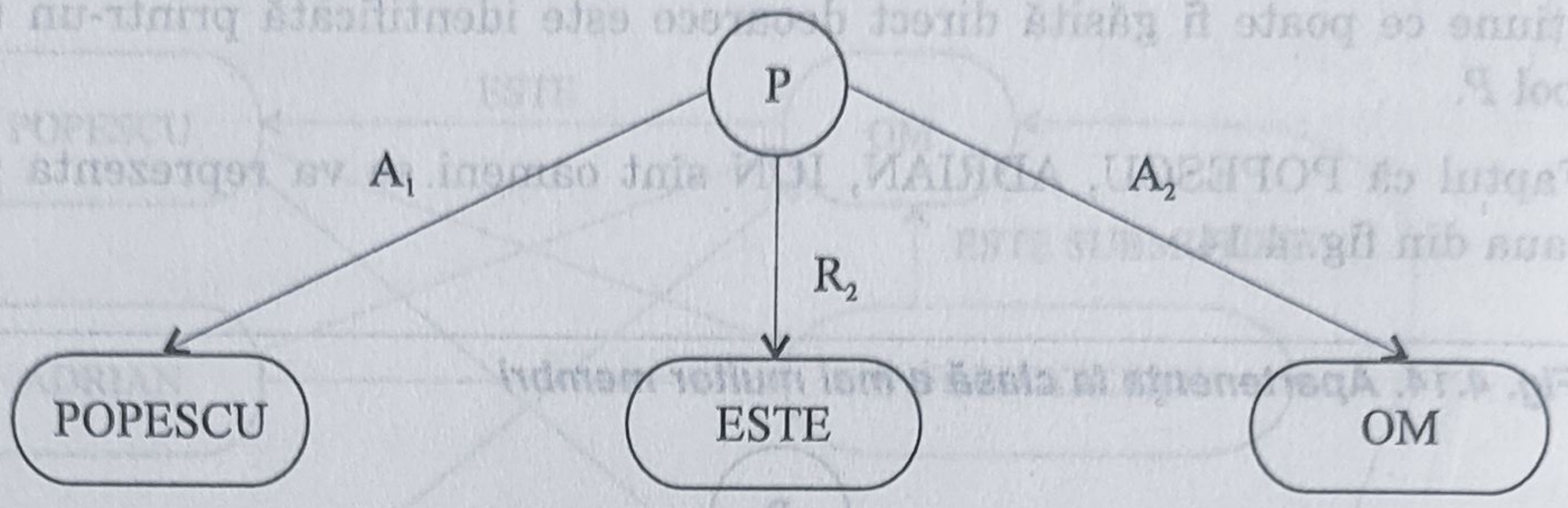
- *R2* relaţie binară;
- A1 specifica termenul aflat în primul loc al relaţiei;
- *A2* specifica termenul aflat pe locul doi al relaţiei.

Dacă se reprezintă aceeaşi aserţiune "Popescu este om" se obţine reţeaua semantică din fig. 4.15:

Cu ajutorul primitivelor taxonomice este asigurată moştenirea proprietăţilor conceptului de către instanţele sale. Spre exemplu, proprietatea conceptului individual AUTOMOBIL de a fi mijloc de transport este moştenită de obiectul D1300 deoarece,

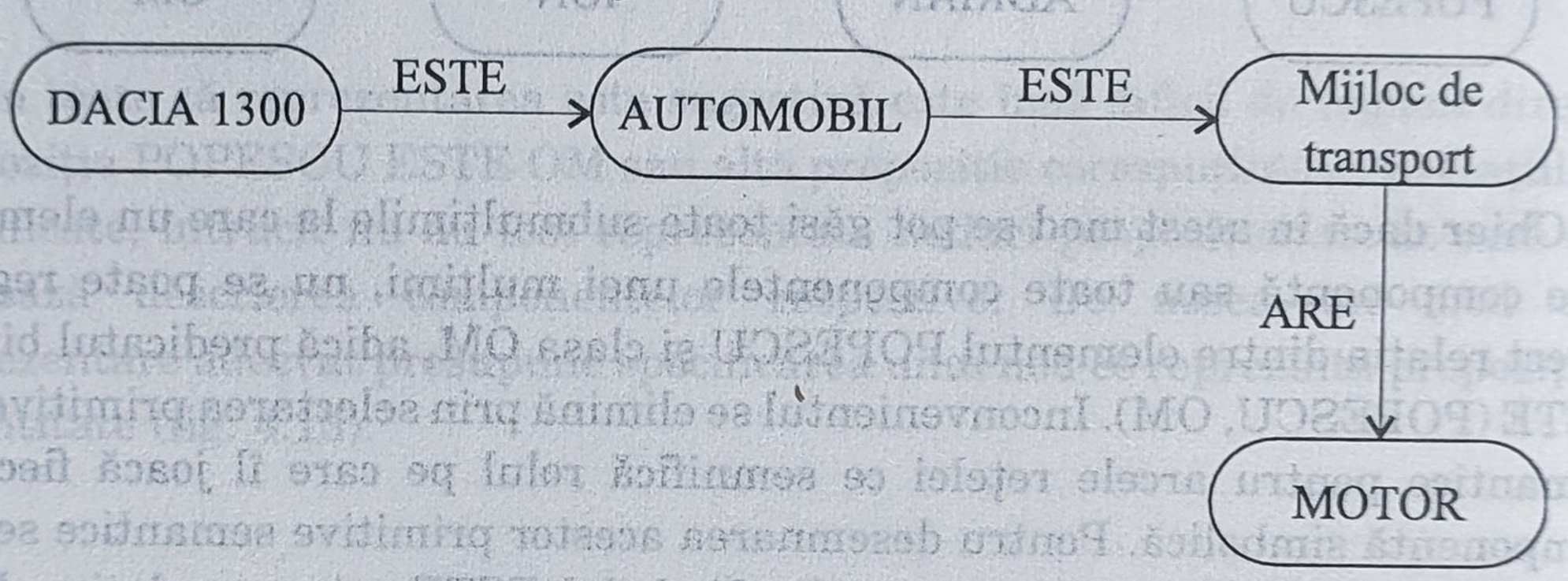
D1300 ESTE AUTOMOBIL
(şi) AUTOMOBIL ESTE MIJLOC DE TRANSPORT
Deci, D1300 ESTE MIJLOC DE TRANSPORT.

▼ ***Fig. 4.15. Reprezentarea unei propoziţii utilizînd primitive semantice cauzale.***



Reţeaua semantică asociată este reprezentată în fig. 4.16.

▼ ***Fig. 4.16. Moştenirea proprietăţilor***



Proprietatea conceptului de mijloc de transport, de a avea motor se transferă şi asupra conceptului individual AUTOMOBIL cît şi la descendenţii săi extensionali adică, D1300 care ARE MOTOR.

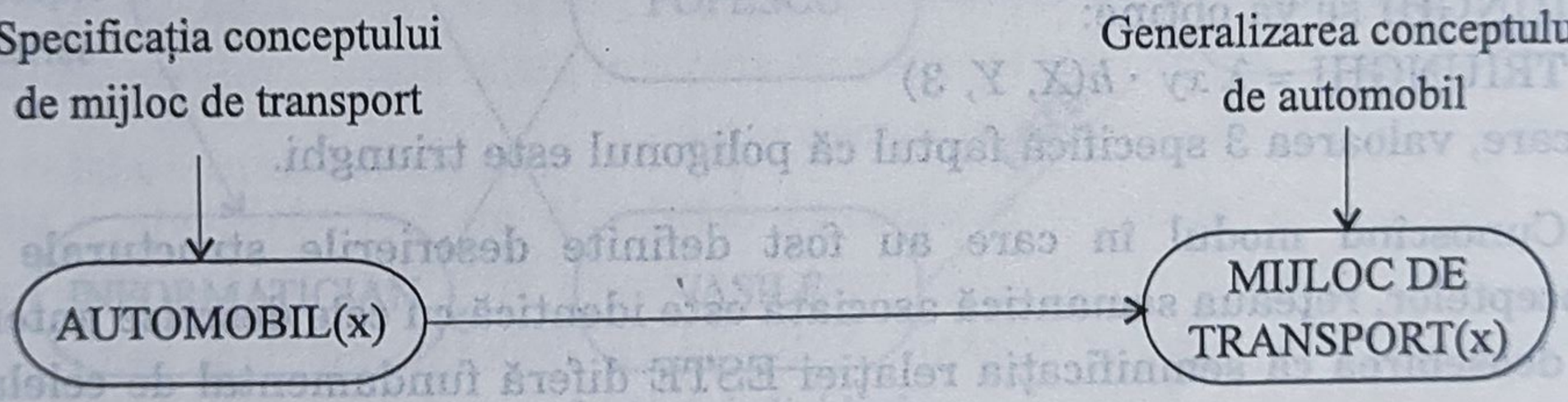
Orice nouă proprietate adăugată ulterior conceptului mijloc de transport se va transmite tuturor descendenţilor extensionali ai acestuia. De exemplu adăugarea proprietăţii că mijloacele de transport au sistem de rulare va fi transmisă tuturor automobilelor şi în particular pentru D1300. Este necesar a se aminti că relaţiile taxonomice se regăsesc în orice reţea semantică, şi au inconvenientul folosirii abuzive a singurei primitive semantice ESTE. Relaţiile taxonomice pot fi stabilite între următoarele tipuri de noduri: concepte generice, concepte individuale şi indivizi. În situaţia în care nodurile sînt concepte generice de tipul mulţime, relaţia taxonomică prin primitiva semantică ESTE semnifică incluziunea determinată de un concept generic la mulţimea determinată de alt concept generic indicînd faptul că orice element

al submulţimii este în acelaşi timp şi element al mulţimii şi că toate proprietăţile conceptului generic de nivel superior se moştenesc de cele de nivel inferior mai puţin acele proprietăţi ce sînt implicate în procesul de constituire a submulţimii.

Pentru conceptul generic AUTOMOBIL se pot instanţia submulţimi ale acestuia funcţie de tipul motorului cu care este echipat. Dintre aceste submulţimi se pot aminti OTTO, DIESEL, WANKEL, ELECTRIC, TURBOPROPULSOR. În anumite situaţii criteriul de grupare a elemetelor în submulţime nu se regăseşte printre proprietăţile definitorii ale conceptului generic ce fixează mulţimea. În aceste cazuri conceptul generic la nivelul submulţimii moşteneşte toate proprietăţile conceptului generic la nivelul mulţimii.

Un mod de stabilire a unei relaţii taxonomice între concepte este cel prin predicate. Pentru exemplul de mai sus predicatul AUTOMOBIL (x), predicat ce semnifică proprietatea obiectului x de a fi automobil este în relaţie taxonomică cu predicatul *MIJLOC DE TRANSPORT* (x) ce specifică proprietatea obiectului x de a fi mijloc de transport. Relaţia între predicate este ilustrată în fig. 4.17.

▼ ***Fig. 4.17. Relaţie taxonomică între predicate***



Se obişnuieşte să se reprezinte în reţele semantice forma predicativă a conceptului fără a se indica variabilele formale ataşate. Desigur că o astfel de reprezentare poate genera confuzii referitoare la acceptitudinea în care trebuiesc reprezentate conceptele.

Formula ce reprezintă o generalizare a predicatului AUTOMOBIL (x) prin predicatul MIJLOC DE TRANSPORT (x) este:

$$\forall x \text{ AUTOMOBIL } (x) \rightarrow \text{MIJLOC DE TRANSPORT } (x)$$

Se mai spune că predicatul AUTOMOBIL (x) este o specializare a predicatului MIJLOC DE TRANSPORT (x). O altă interpretare a relaţiilor taxonomice între concepte generice în reţelele semantice este λ - abstractizarea conceptelor descrise prin variabilele formale. Variabilele formale în expresiile predicative

ajută la desfăşurarea corectă a procesului de substituţie. Cu notaţia uzuală nu se poate distinge cazul în care $f(x)$ se interpretează ca o valoare a funcţiei pentru o valoare nedeterminată a variabilei x, adică o instantă.

Teoria λ - conversiei introduce notaţii distincte pentru interpretarea lui $f(x)$ ca funcţie de x, cu operatorul de abstractizare funcţională $\lambda x \cdot f(x)$ ce arată că funcţia $f(x)$ este de variabilă x cu condiţia ca x să apară liber în $f(x)$. Astfel funcţia de mai multe variabile $g(x, y, z)$ poate fi interpretată astfel:

- funcţie de o singură variabilă
  $\lambda x \cdot g(x, y, z)$
- funcţie de două variabile
  $\lambda xy \cdot g(x, y, z)$
- funcţie de trei variabile
  $\lambda xyz \cdot g(x, y, z)$.

Pentru descrierea generală a conceptului generic poligon se procedează astfel:

POLIGON = $\lambda xy\, n \cdot h(X, Y, n)$ în care

$X = (x1, x2,..., xn)$ şi $Y = (y1, y2,..., yn)$

sînt coordonatele vîrfurilor poligonului şi n numărul de vîrfuri.

Dacă se doreşte dezvoltarea conceptului generic POLIGON spre conceptul TRIUNGHI se va obţine:

TRIUNGHI = $\lambda xy \cdot h(X, Y, 3)$

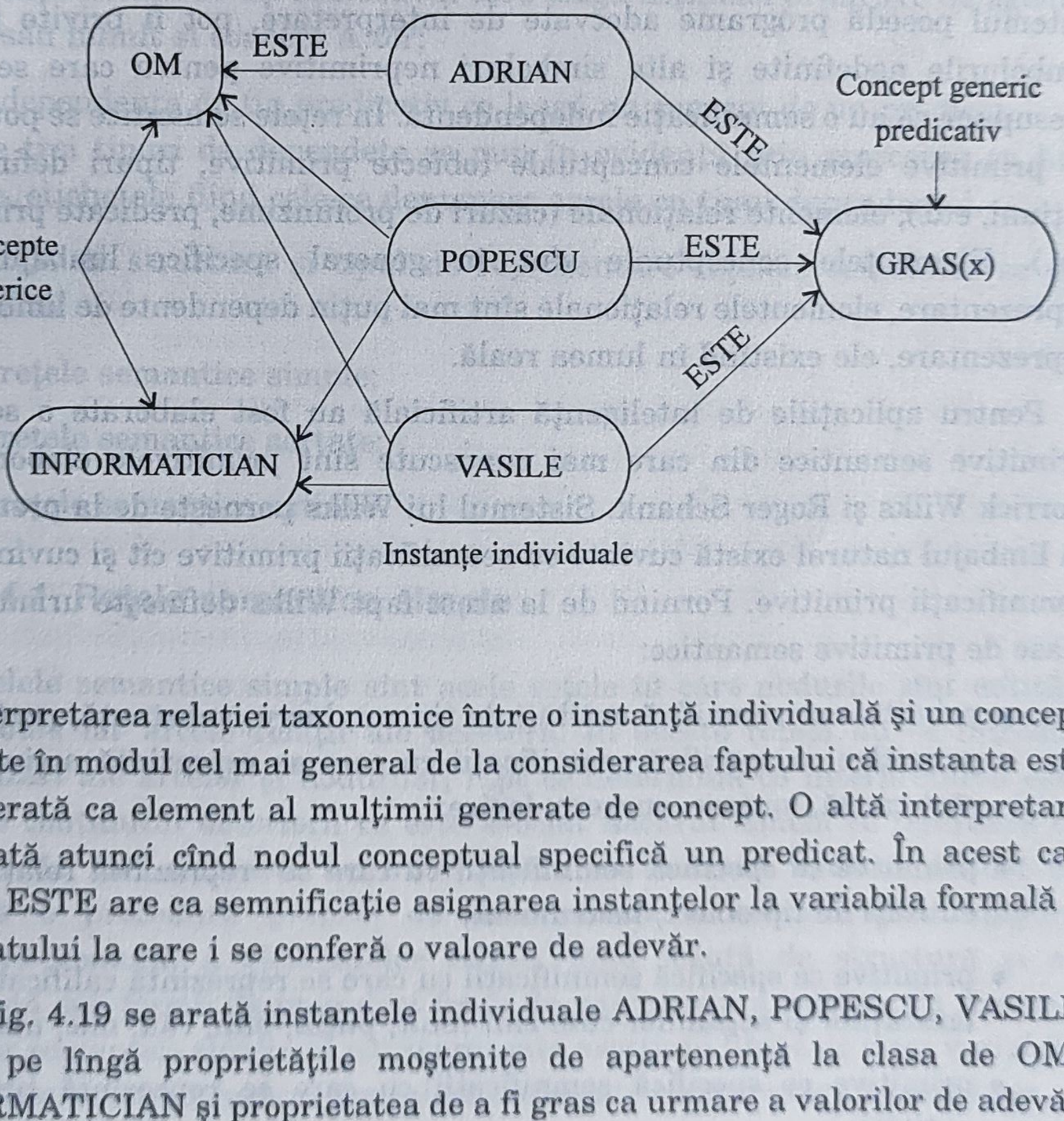
în care, valoarea 3 specifică faptul că poligonul este triunghi.

Cunoscînd modul în care au fost definite descrierile structurale ale conceptelor, reţeaua semantică asociată este identică cu cele descrise anterior cu deosebirea că semnificaţia relaţiei ESTE diferă fundamental de celelalte aserţiuni (fig. 4.18).

▼ ***Fig. 4.18.*** *Relaţie taxonomică cu abstractizare.*

Reţeaua de mai sus ilustrează modul de instanţiere a conceptelor prin restricţionarea valorilor unor variabile ale conceptului generic. Se poate ca abstratizarea să pună în relaţie concepte generice ce se includ unele în altele. Analizînd funcţia $f(x, y)$ se observă două concepte generice $x \cdot f(x, y)$ şi $y \cdot f(x, y)$ ce sînt în relaţia λ - abstractizare a conceptului generic $xy \cdot f(x, y)$. Incluziunea conceptelor apare totdeauna cînd un concept generic de nivel inferior are o descriere structurală proprie. Variabilele formale din abstratizarea conceptului de nivel inferior trebuie să se găsească în abstractizarea de nivel superior. Dacă variabilele formale figurează la nivel superior dar nu sînt precizate în abstractizarea de nivel inferior, ele trebuie să figùreze în partea descriptivă a funcţiei fie ca simboluri, fie ca restricţii ale acestora.

▼ ***Fig. 4.19. Relaţie taxonomică cu predicaţie.***



Interpretarea relaţiei taxonomice între o instanţă individuală şi un concept porneşte în modul cel mai general de la considerarea faptului că instanta este considerată ca element al mulţimii generate de concept. O altă interpretare este dată atunci cînd nodul conceptual specifică un predicat. În acest caz relaţia ESTE are ca semnificaţie asignarea instanţelor la variabila formală a predicatului la care i se conferă o valoare de adevăr.

În fig. 4.19 se arată instantele individuale ADRIAN, POPESCU, VASILE ce au pe lîngă proprietăţile moştenite de apartenenţă la clasa de OM, INFORMATICIAN şi proprietatea de a fi gras ca urmare a valorilor de adevăr

date de instantele individuale predicatului GRAS (x). λ - abstractizarea conceptuală poate fi cunoscută pînă la nivelul constantelor sistemului cognitiv, adică la instanţele situate la nivel infim.

A fost reprezentată pînă în prezent cea mai răspîndită primitivă semantică, relaţia taxonomică, relaţie ce nu epuizează toate posibilităţile pe care primitiva le oferă reprezentării cunoaşterii. Desigur că, nu orice cunoaştere se poate exprima printr-o singură primitivă semantică cu un singur simbol relaţional. Instrumentul cel mai performant de reprezentare a cunoaşterii este limbajul natural ce dispune de un vocabular vast în continuă evoluţie. Dacă prin simboluri de bază înţelegem literele, numărul simbolurilor este limitat, însă combinarea simbolurilor de bază determină diverse simboluri semantice. Problema primitivelor semantice este destul de controversată datorită inexistenţei unei definiţii unanim acceptate pentru această categorie. Dacă se consideră primitivele semantice ca simboluri nedefinite în limbaj cu semnificaţia cunoscută de utilizator, pentru care sistemul posedă programe adecvate de interpretare, pot fi privite printre simbolurile nedefinite şi alte simboluri neprimitive pentru care se poate presupune că au o semnificaţie independentă. În reţele semantice se pot defini ca primitive elementele conceptuale (obiecte primitive, tipuri definite de acţiuni, etc.), elemente relaţionale (cazuri de profunzime, predicate primitive, etc.). Elementele conceptuale sînt în general specifice limbajului de reprezentare, elementele relaţionale sînt mai puţin dependente de limbajul de reprezentare, ele existînd în lumea reală.

Pentru aplicaţiile de inteligenţă artificială au fost elaborate o serie de primitive semantice din care mai cunoscute sînt primitivele elaborate de Yorrick Wilks şi Roger Schank. Sistemul lui Wilks porneşte de la premisa că în limbajul natural există cuvinte cu semnificaţii primitive cît şi cuvinte fără semnificaţii primitive. Pornind de la acest fapt Wilks defineşte următoarele clase de primitive semantice:

- primitive ce semnifică entităţi de tip om, lucru, substanţă, parte;
- primitive ce specifică semnificaţii cu care se reprezintă acţiuni cum sînt cauză, mutare, curgere, lovire;
- primitive ce specifică semnificaţii cu care se reprezintă relaţii între entităţi de tip obiect, instrument;
- primitive ce specifică semnificaţii cu care se reprezintă calificative ale entităţilor şi acţiunilor cum sînt mult, puţin, bun, rău, mic, mare;
- primitive ce specifică semnificaţii cu care se reprezintă tipuri de entităţi;

- primitive ce specifică clase de primitive, obiecte fizice, oameni, obiecte în interior, exterior.

Pentru cuvintele ce nu sînt primitive se dau definiţii cu ajutorul primitivelor semantice. Primitivele semantice trebuie să fie independente în sensul că nu se pot defini cu ajutorul altor primitive semantice asigurînd astfel consistenţa, în sensul că nu sînt definite circular.

Sistemul lui Schank sau teoria dependenţelor conceptuale foloseşte reprezentarea fără ambiguităţi a semnificaţiei utilizînd 11 acţiuni elementare grupate în categorii. Într-o astfel de reţea semantică conceptele sînt legate prin relaţii numite dependente, ce se disting prin tip şi etichetă. Pot exista mai multe tipuri de dependenţe cum sînt:

- dependenţă de tip cauzal materializată prin descrierea relaţiilor conceptuale ale obiectului şi legătura sa cu actul primitiv;
- dependenţă de tip subiectival care leagă acţiunea primitivă de agentul său numit şi concept actor;
- dependenţa de tip predicativ ce leagă un concept de un predicat.

Cele trei tipuri de dependeţe se pun în evidenţă prin marcarea cu linii diferite, etichetele fiind cele ce denumesc arcele cu tipul dependenţei.

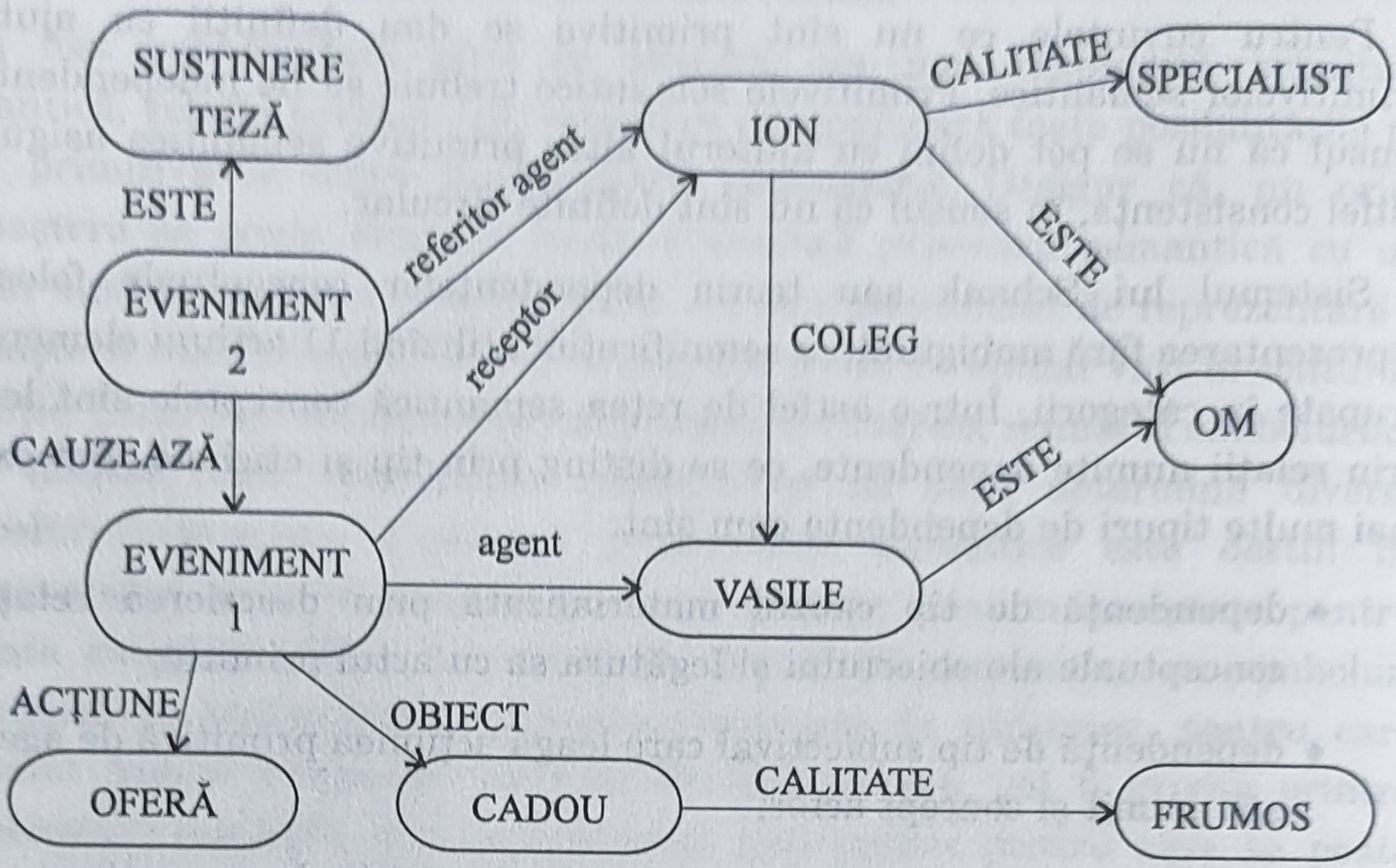
Funcţie de structură şi tipul de reprezentare reţelele semantice se pot clasifica în:

- reţele semantice simple;
- reţele semantice sortate;
- reţele semantice extinse.

4.2.4.1. Reţele semantice simple

Reţelele semantice simple sînt acele reţele în care nodurile sînt entităţi individuale iar arcele relaţii ale acestora. În aceste reţele nu se introduc specializări ale arcelor şi nodurilor, fapt ce determină că interpretarea este dată de conţinutul descrierii ce este asociat fiecărui simbol ce figurează ca etichetă de nod sau arc. La interpretare se asociază fiecărui simbol de etichetă o procedură proprie de determinare a semnificaţiei, singura trăsătură comună a elementelor reţelei fiind legată de structură şi se manifestă sub formă de proceduri fizice de parcurgere a reţelei. Cu ajutorul reţelelor semantice simple se pot reprezenta aserţiuni libere de orice variabilă întrucît nu se admit noduri specializate aferente reprezentării variabilelor.

▼ ***Fig. 4.20. Reţea semantică simplă***



Se presupune reprezentarea piesei de cunoaştere ce specifică faptul că *"la susţinerea tezei de doctorat de către Ion care este un bun specialist, colegul său Vasile îi oferă un frumos cadou"*. Se obţine o reţea semantică ce are structura din fig. 4.20.

Se observă din studiul reţelei semantice de mai sus că în această reprezentare aserţiunea nu este realizată în forma propoziţională. Se cunoaşte faptul că Ion şi Vasile sînt oameni adică aparţin clasei oamenilor. Acţiunea de oferire a cadoului de către Vasile este un eveniment pentru care Vasile este agent şi Ion este receptor. Pentru Ion susţinerea tezei de doctorat este de asemenea un eveniment. Acest eveniment este în relaţie cauzală cu evenimentul oferirii cadoului. Reţelele semantice simple nu oferă noduri preferenţiale în iniţierea procesului de interpretare, selecţia unui nod ca nod iniţial este determinată de aplicaţia pentru care este construită baza de cunoştinţe. Candidaţii pentru poziţia de nod iniţial sînt selectaţi după diverse criterii dintre care cele mai întîlnite sînt:

- nodul etichetat cu un simbol identic numelui piesei de cunoaştere;
- selecţia candidaţilor dintre nodurile ce au proprietatea de a fi noduri sursă pentru toate relaţiile în care sînt implicate.

Pentru descrierea unei reţele semantice simple aceasta este descompusă în triplete de forma

(<*nod_sursă*> <*relaţie*> <*nod_destinaţie*>).

Descompunerea sugerează utilizarea de liste în LISP. În reţeaua semantică din fig. 4.20 singurul nod ce este numai nod sursă este nodul <Eveniment_2> ce poate fi considerat nod iniţial, lista pornind descrierea de la acest nod este:

```
(Eveniment_2 este susţinerea tezei)
(Eveniment_2 referitor Ion)
  (Ion este om)
  (Ion calitate specialist)
  (Ion coleg Vasile)
    (Vasile este om)
(Eveniment_2 cauzează eveniment_1)
  (Eveniment_1 acţiune oferă)
  (Eveniment_1 obiect cadou)
     (Cadou calitate frumos)
  (Eveniment_1 receptor Ion)
  (Eveniment_1 agent Vasile).
```

O altă modalitate de reprezentare a reţelei constă în construirea conceptului pentru fiecare nod sursă şi asocierea acestuia cu perechi (<relaţie> <nod destinaţie>).

```
((Eveniment_2 (este susţinerea tezei)
        (referitor Ion)
        (cauzează eveniment_1))
(Ion   (este om)
      (calitate specialist)
      (coleg Vasile))
(Vasile (este om))
((Eveniment_1 (acţiune oferă)
          (obiect cadou)
          (receptor Ion)
          (agent Vasile)
((Cadou (calitate frumos)).
```

Metoda are avantajul eliminării redundanţelor ce apar la specificarea nodului sursă în toate tripletele ce descriu relaţiile sale cu nodurile destinaţie.

4.2.4.2. Reţele semantice sortate

Acest tip de reţele sînt caracterizate prin faptul că folosesc o specializare a nodurilor şi arcelor indicînd prin aceasta diferenţe de natură structurală ce nu pot fi rezolvate prin etichetare. Cel mai simplu tip de etichetare este cel prin care se specifică un nod de tip propoziţie. În elaborarea tehnicilor de etichetare a nodurilor şi tipurilor de relaţii se cunosc mai multe abordări dintre care mult utilizată este cea definită de Fillmore. Fillmore stabileşte un număr restrîns de relaţii primitive ce pot acoperi o varietate semantică largă. O parte dintre acestea sînt:

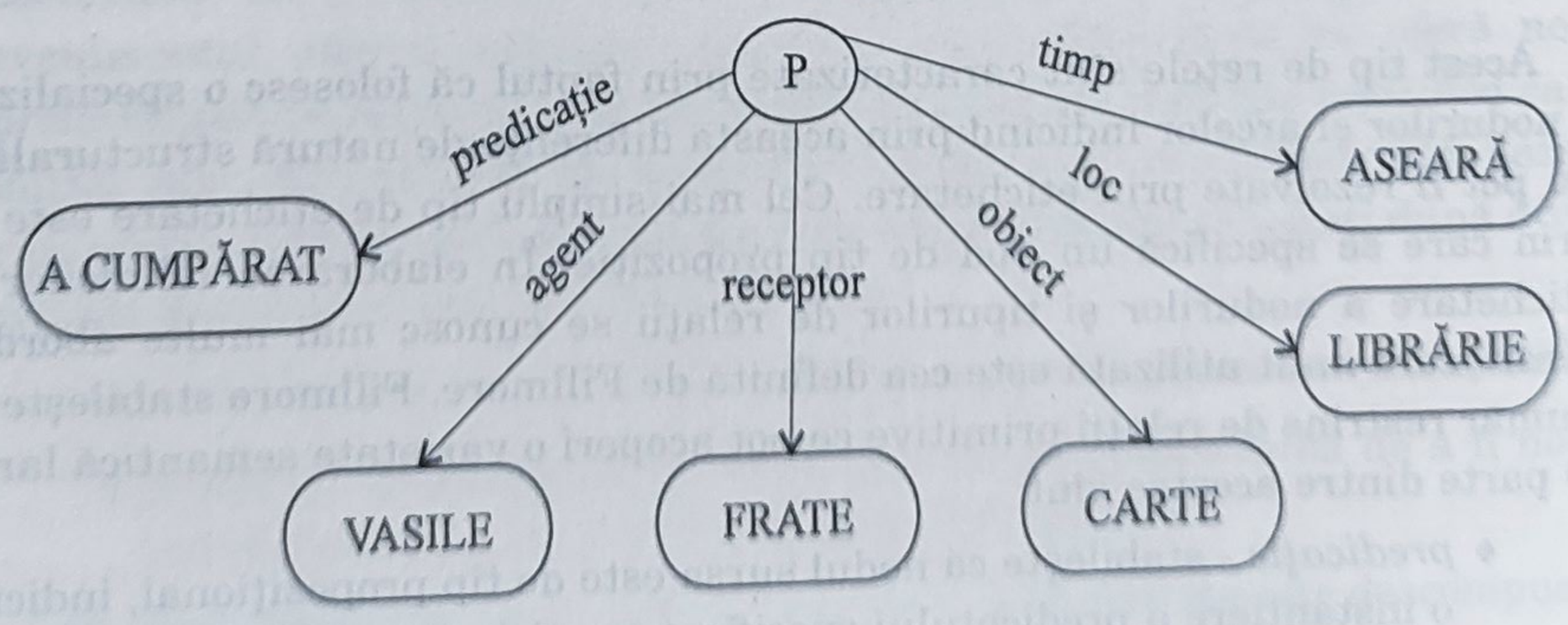
- *predicaţie* - stabileşte că nodul sursa este de tip propoziţional, indicînd o instanţiere a predicatului specificat în eticheta nodului destinaţie ce reprezintă o instanţiere sau o stare (echivalentă cu verbul într-o propoziţie);

- *agent* - are ca nod destinaţie o entitate cu capacitatea de a efectua acţiunea. Entitatea este numită şi "actor" şi are rolul de subiect logic;
- *receptor* - indică un nod destinaţie ce are capacitatea de a participa la acţiune;
- *obiect* - indică ca nod destinaţie o entitate ce este afectată de acţiunea specificată pentru propoziţia din nodul sursa;
- *instrument* - are ca nod destinaţie o entitate ce este folosită de agent la efectuarea acţiunii suportate de obiect;
- *loc* - cu nod de destinaţie o entitate exprimînd o propoziţie ce indică locul desfăşurării acţiunii;
- *timp* - indică un nod destinaţie ce specifică timpul sub o anumită formă. Poate fi exprimată durata prin două relaţii momentane (moment_iniţial, moment_final), fie ca o relaţie durată ce arată valoarea intervalului fără a se specifica scara timpului.

Se consideră ca exemplu piesa de cunoaştere *aseară Vasile a cumpărat de la librărie o carte pentru fratele său,* ce se reprezintă sub forma propoziţională într-o reţea semantică sortată ca în fig. 4.21, în care se folosesc relaţiile cauzale menţionate.

Predicaţia este reprezentată cu ajutorul unui simbol predicativ, cu o aritate corespunzătoare descripţiei, cu argumentele plasate în două clase şi anume *argumente de bază* obligatorii pentru a defini o instantă valabilă a predicatului şi *argumente periferice* sau opţionale în instanţieri ce aduc în reţea relaţii suplimentare prin care se conturează cunoaşterea sub aspecte ce nu depind exclusiv de predicaţia considerată.

▼ ***Fig. 4.21. Reprezentarea propoziţională.***



Fie $P((x_1, x_2,..., x_p), (y_1, y_2,..., y_q))$ expresia predicativă generatoare a propoziţiilor ce sînt formate cu un verb. O predicaţie validă are aritatea n cu $p <= n <= p + q$. Expresia $P(x_1, x_2,..., x_p)$ se mai numeşte şi predicaţie nucleară iar expresia $P((x_1, x_2,..., x_p), (y_1, y_2,..., y_q))$ se mai numeşte şi predicaţie extinsă.

Se dau mai jos cîteva sorturi de expresii predicative:

- sortul expresiilor predicative verbale (*V*) ce apar în instante ca predicate ale propoziţiilor;
- sortul expresiilor predicative adjectivale (*A*) ce apar în instante ca modificatori de termeni;
- sortul expresiilor predicative nominale (*N*) ce apar în instante ca definiţii de termeni.

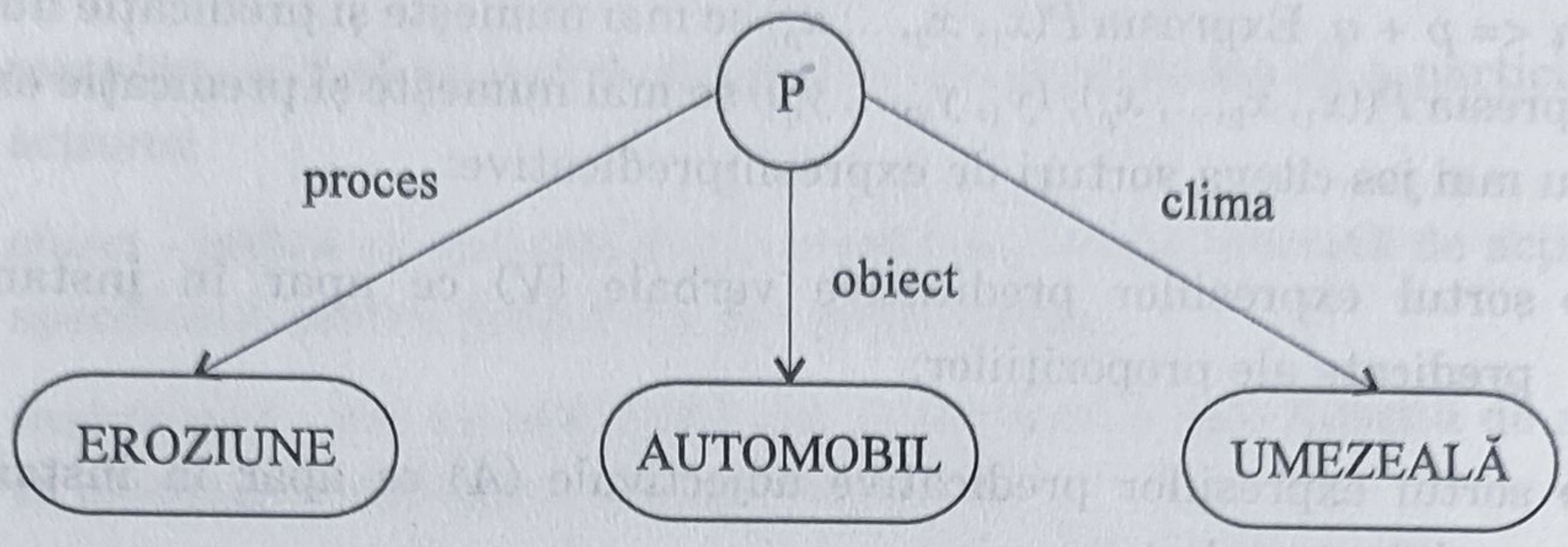
Cadrul definitoriu pentru predicate cuprinde următoarele date iniţiale:

- formatul predicatului compus din numele identificator, lista argumentelor de bază şi a argumentelor periferice;
- sortul predicatului;
- funcţia semantică a argumentelor exprimate sub forma relaţiilor cauzale;
- restricţii de selecţie a termenilor ce candidează pe poziţia de argumente ale predicatului.

Starea de lucruri descrisă de o predicaţie dă o valoare de adevăr predicatului. Stările de lucruri ce nu se modifică în timp se numesc statice spre deosebire de cele dinamice ce sînt caracterizate de schimbări în timp. Dacă în cadrul predicatului există un argument numit controlor ce are proprietatea de a determina starea de lucruri asociată predicaţiei se spune că starea este controlată. Dinamismul şi controlul determină noi sorturi ale predicaţiei şi anume:

- *acţiunea* - stare dinamică şi controlată;
- *procesul* - corespunde unei stări de lucruri dinamice şi necontrolate;
- *poziţia* - corespunde unei stări de lucruri statice şi controlate;
- *situaţia* - corespunde unei stări de lucruri statice dar necontrolate.

▼ **Fig. 4.22. Reprezentarea unei predicaţii din sortul acţiune**



Pentru predicatele din sortul acţiune, relaţiile cauzale de bază determină argumentele părţii nucleare cum sînt agentul şi obiectul. Relaţiile cauzale periferice specifică receptorul acţiunii, direcţia desfăşurării acţiunii, instrumentul folosit, sursa obiectelor acţiunii, locul şi timpul desfăşurării. Predicatele din sortul proces nu au relaţie cauzală de tip agent întrucît sortul este necontrolat, singura relaţie cauzală de bază este obiectul afectat. Relaţiile periferice sînt cele ce specifică relaţii cauzale ce apar la sortul acţiune. Spre exemplu procesul eroziunii automobilului de către umezeală (fig. 4.22), cu toate că există o cauză a procesului de eroziune, asupra umezelii nu se exercită nici un control, motiv pentru care a fost menţionată în relaţia cauzală clima ca fiind cea care produce acest proces.

Predicatele din sortul poziţie posedă un agent ce controlează poziţia, şi se va nota cu termenul poziţionator. Cu o relaţie de acest tip se pot reprezenta stări de lucruri în care poziţionatorul se poziţionează pe sine.

Relaţiile cauzale se pot extinde fără restricţii putînd fi definite următoarele tipuri de relaţii:

- ♦ relaţii ce extind specificarea stării de lucruri: calitate, modalitate;
- ♦ relaţii ce referă predicaţia la alţi participanţi: iniţiator, cunoscător, executor, terminator;
- ♦ relaţii ce extind dimensiunile spaţiale: sub, lîngă, deasupra;
- ♦ relaţii ce extind dimensiunile temporare: durata, frecvenţa;
- ♦ relaţii de interpoziţionare între o stare şi alte stări: cauză, motiv, scop, rezultat, circumstanţă.

Sorturile analizate fac posibilă reprezentarea cunoaşterii factuale ce implică reprezentarea obiectelor a căror complexitate este limitată la reprezentarea propoziţiilor simple. Pentru cunoaşterea conceptuală ce implică obiecte abstracte cum sînt variabilele, se introduce un nou sort de nod

corespunzător acestor simboluri. Cu aceste noduri se reprezintă obiecte conceptuale participante la aserţiuni ce îndeplinesc şi rolul de conectare a propoziţiilor. Un exemplu de acest tip apare în reprezentarea piesei de cunoaştere

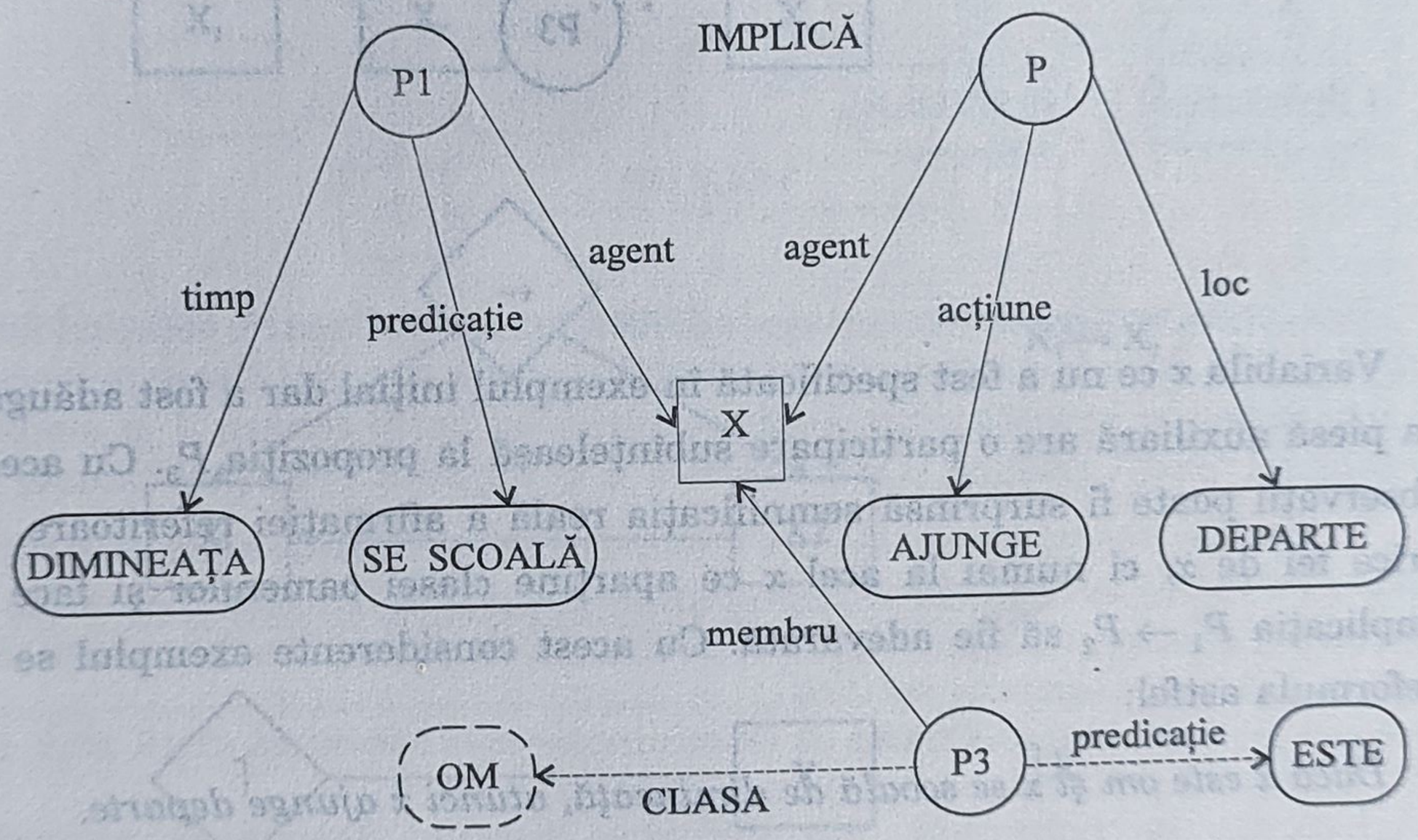
cine se scoală de dimineaţă, departe ajunge.

În această propoziţie cine este echivalent cu orice om, expresie ce se poate scrie cu formula:

$\forall x$ (*este* (x, *om*) $\wedge$ *se_scoală* (x, *dimineaţă*) $\rightarrow$ *ajunge* (x, *departe*).

Reprezentarea acestei formule în reţeaua semantică, presupune reprezentarea relaţiei taxonomice *x este om* ce introduce relaţiile cauzale de membru şi clasa, şi au aceleaşi funcţii semantice ca şi relaţiile cunoscute *agent* şi *obiect*, atunci cînd se specifică o apartenenţă la clasă. Pentru această formulă se obţine reţeaua semantică din fig. 4.23.

▼ ***Fig. 4.23. Conectarea reţelelor semantice prin intermediul nodurilor de variabile***

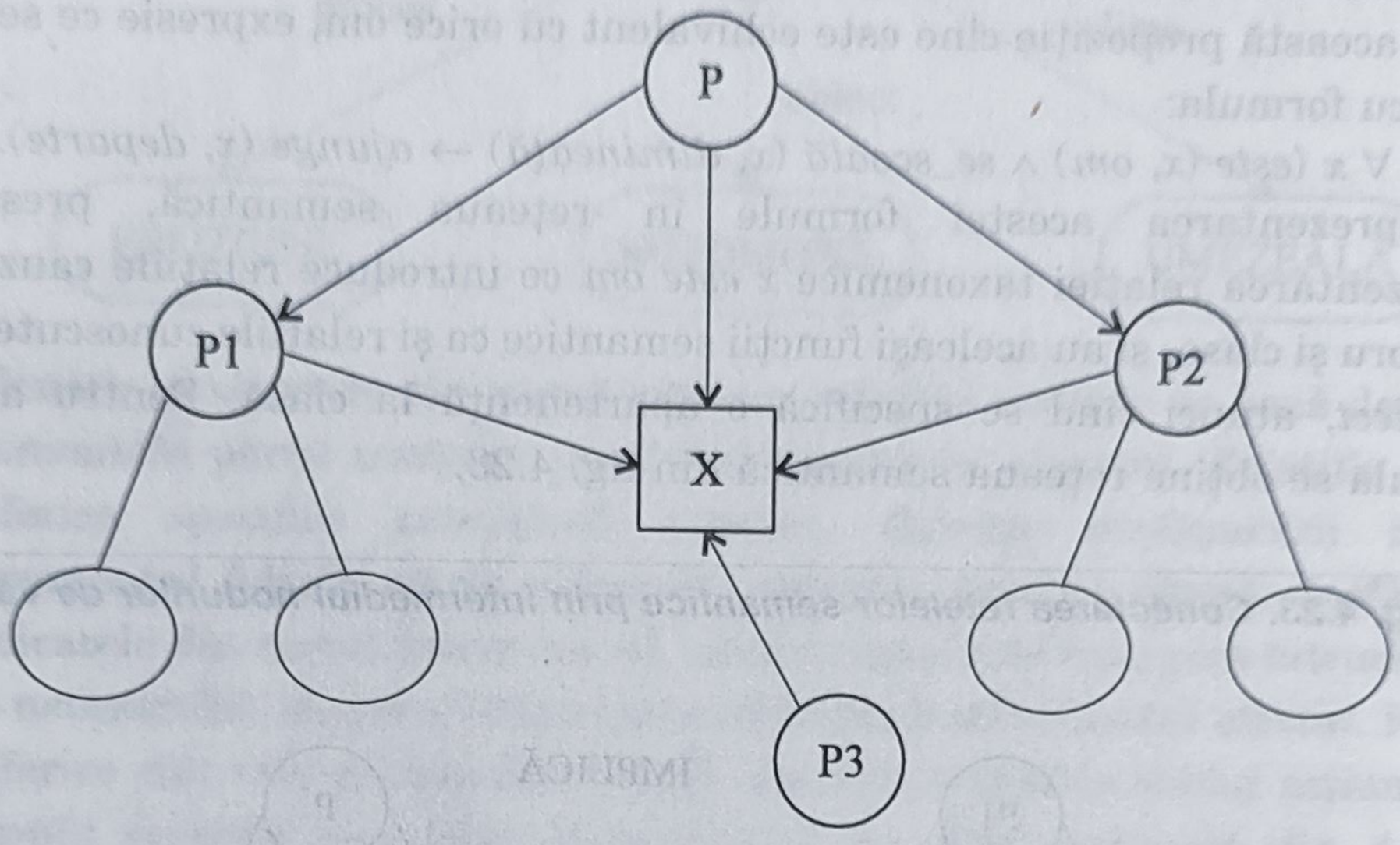


În această reţea s-a dat o rezolvare implicită introducerii cuantificatorilor, prin faptul că expresia reprezentată utilizează un singur cuantificator universal. Dacă reţeaua are o dimensiune mai mare, se introduc cuantificatorii explicit. Implicaţia poate fi interpretată nu numai ca o relaţie interpropoziţională ci şi ca o propoziţie de tipul

P = *oricare* (x) *dacă* ($P_1(x)$) *atunci* ($P_2(x)$)

ce conduce la reprezentarea din fig. 4.24. în care se reprezintă doar propoziţiile şi cuantificatorul.

▼ **Fig. 4.24. Utilizarea explicită a cuantificatorului universal**

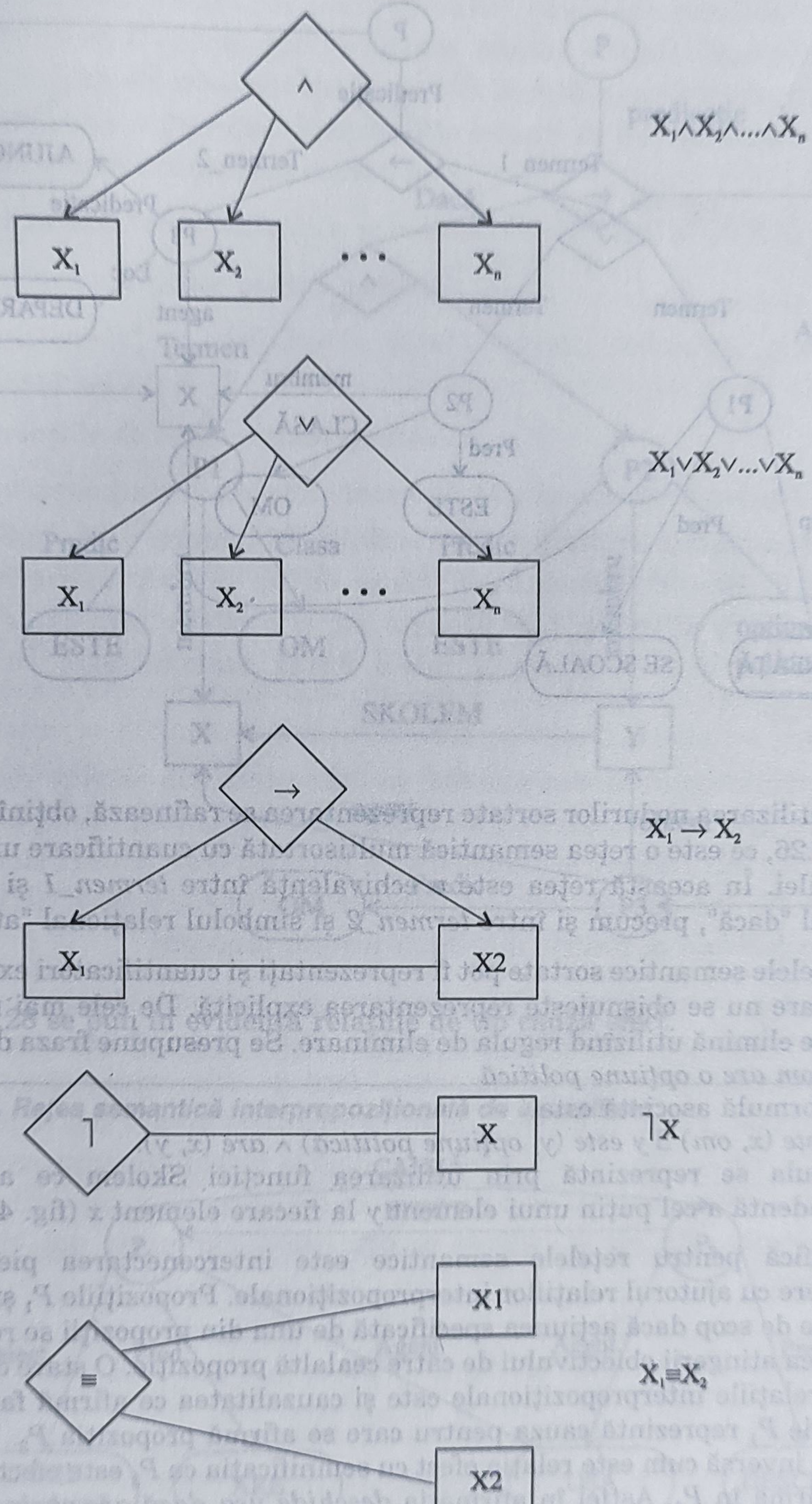


Variabila x ce nu a fost specificată în exemplul iniţial dar a fost adăugată ca piesă auxiliară are o participare subînţeleasă la propoziţia P_3. Cu aceste observaţii poate fi surprinsă semnificaţia reală a afirmaţiei referitoare, nu orice fel de x, ci numai la acel x ce aparţine clasei oamenilor şi face ca implicaţia $P_1 \rightarrow P_2$ să fie adevărată. Cu acest considerente exemplul se va reformula astfel:

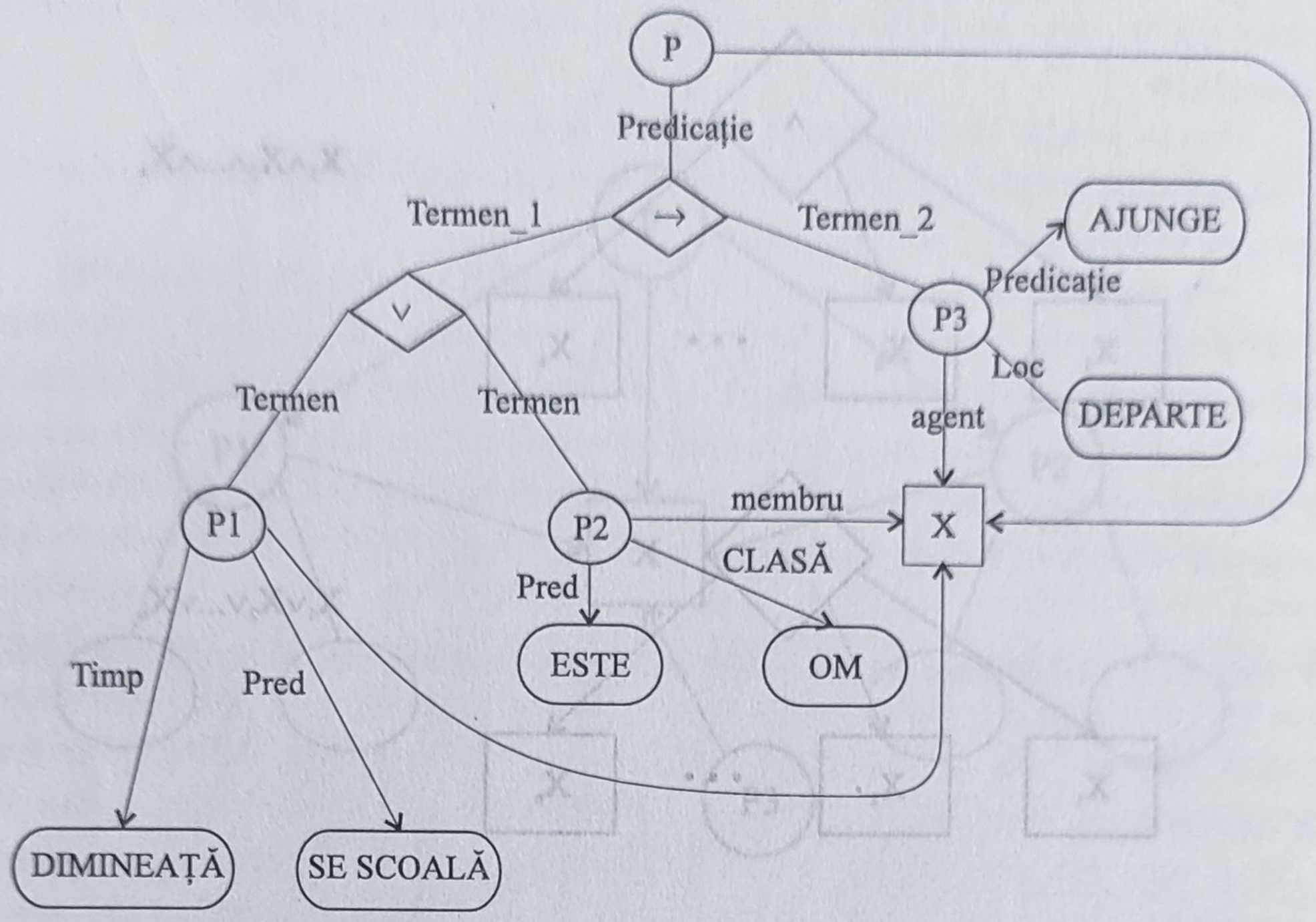
Dacă x este om şi x se scoală de dimineaţă, atunci x ajunge departe.

Reprezentarea frazei într-o reţea semantică necesită un instrument adecvat pentru reprezentarea conectivei ŞI, a conectivei SAU, şi a altor conective logice. În acest scop se utilizează sorturi de noduri etichetate drept conective logice, noduri ce sînt în relaţie cu alte noduri ce reprezintă termeni ai unei formule pe care o determină. Nodurile etichetate ca sorturi de conective logice sînt reprezentate prin romburi şi sînt prezentate în fig. 4.25.

▼ *Fig. 4.25. Reprezentarea nodurilor specializate*



▼ ***Fig. 4.26. Reţea sortată cu cuantificare universală***



Prin utilizarea nodurilor sortate reprezentarea se rafinează, obţinînd forma din fig. 4.26, ce este o reţea semantică multisortată cu cuantificare universală a variabilei. În această reţea este o echivalenţă între *termen_1* şi simbolul relaţional "dacă", precum şi între *termen_2* şi simbolul relaţional "atunci".

În reţelele semantice sortate pot fi reprezentaţi şi cuantificatori existenţiali pentru care nu se obişnuieşte reprezentarea explicită. De cele mai multe ori aceştia se elimină utilizînd regula de eliminare. Se presupune fraza de mai jos

orice om are o opţiune politică

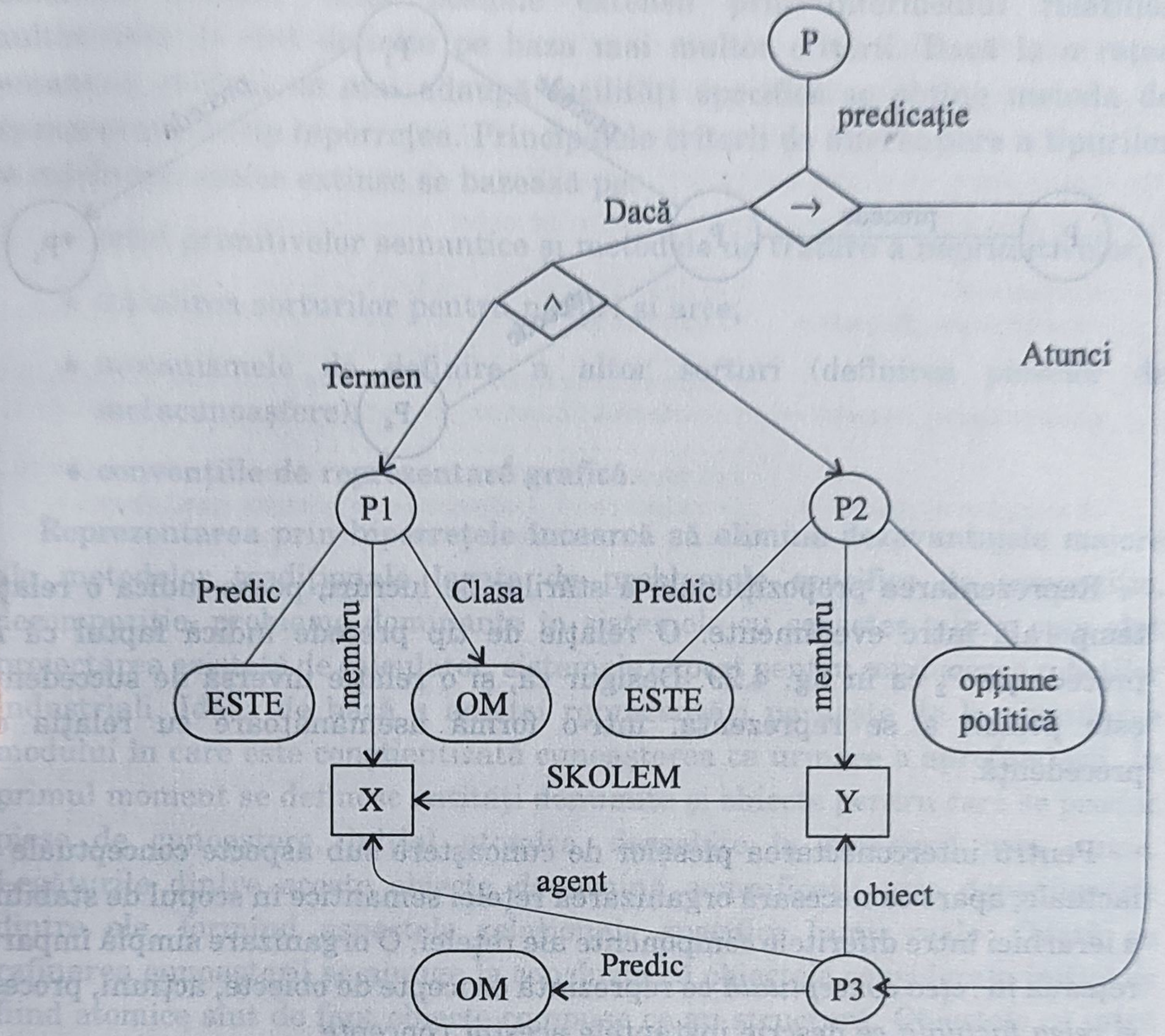
a cărei formulă asociată este

$\forall x$ *este* $(x, om)\ \exists y$ *este* $(y,$ *opţiune politică*$) \wedge$ *are* (x, y).

Formula se reprezintă prin utilizarea funcţiei Skolem ce asigură o corespondentă a cel puţin unui element y la fiecare element x (fig. 4.27).

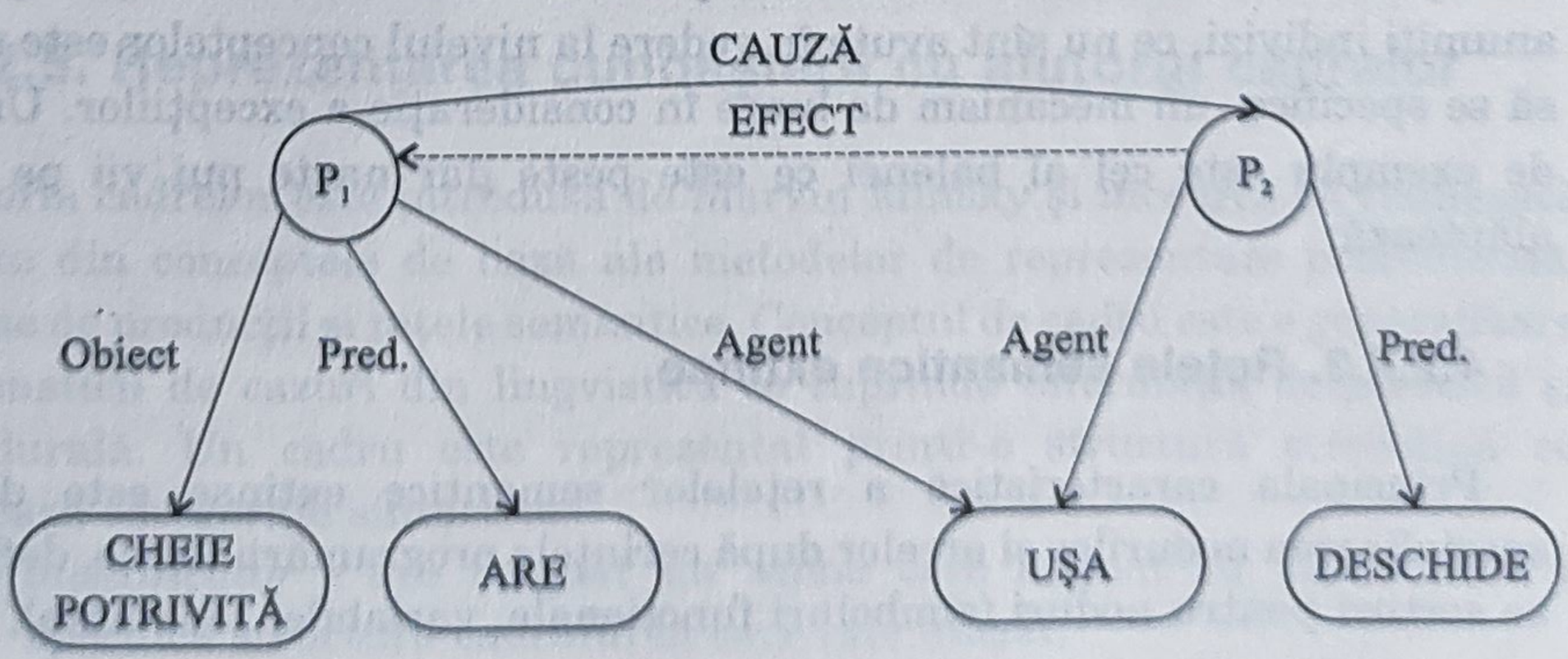
Specifică pentru reţelele semantice este interconectarea pieselor de cunoaştere cu ajutorul relaţiilor interpropoziţionale. Propoziţiile P_1 şi P_2 pot fi în relaţie de scop dacă acţiunea specificată de una din propoziţii se realizează în vederea atingerii obiectivului de către cealaltă propoziţie. O stare de lucruri pentru relaţiile interpropoziţionale este şi cauzalitatea ce afirmă faptul că o propoziţie P_1 reprezintă cauza pentru care se afirmă propoziţia P_2. Există şi o relaţie inversă cum este relaţia efect cu semnificaţia ca P_2 este efectul a ceea ce se afirmă în P_1. Astfel în afirmaţia *deschide uşa dacă are cheie potrivită* reprezentarea printr-o reţea semantică pune în evidenţă relaţia cauza efect.

▼ ***Fig. 4.27. Reţea sortată cu cuantificare existenţială***

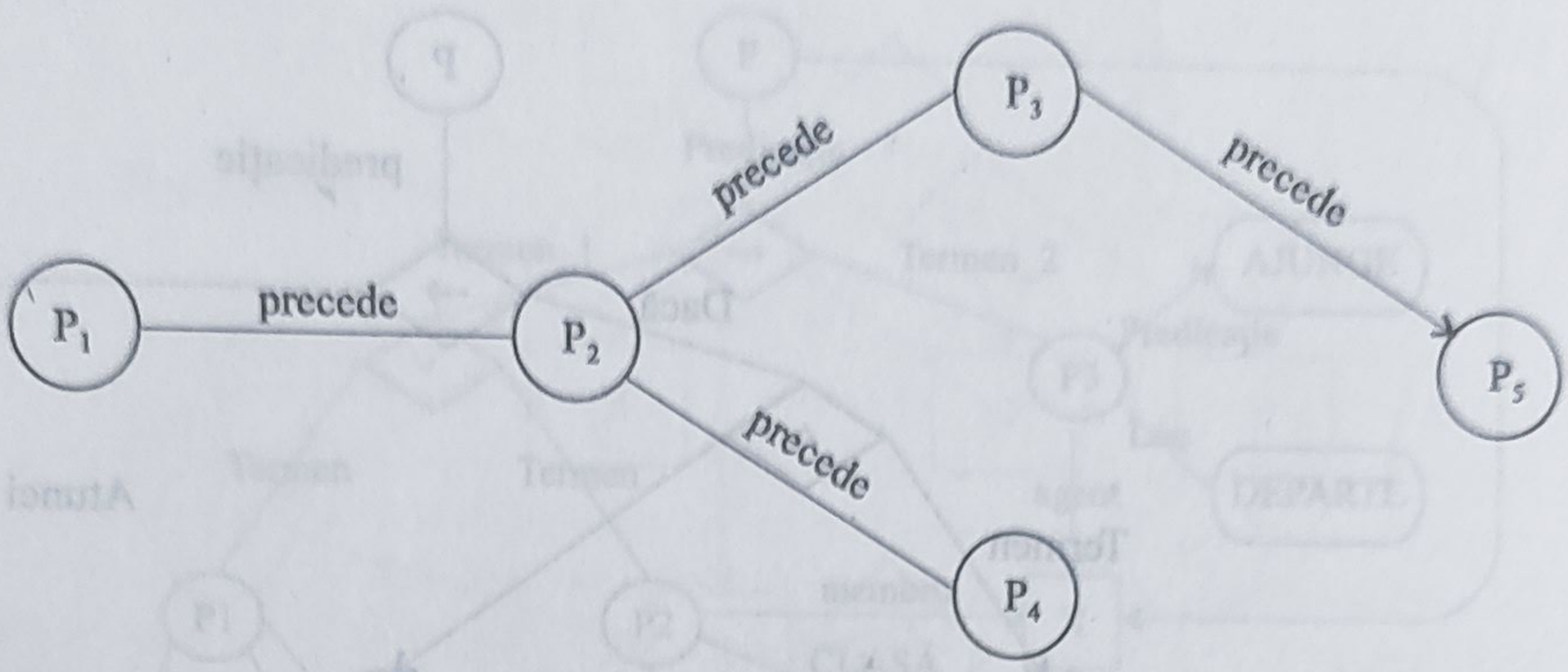


În fig. 4.28 se pun în evidenţă relaţiile de tip cauză efect.

▼ ***Fig. 4.28. Reţea semantică interpropoziţională de cauzalitate***



▼ ***Fig. 4.29. Depedenţa temporală***



Reprezentarea propoziţională a stărilor de lucruri, poate indica o relaţie temporală între evenimente. O relaţie de tip precede indică faptul că P_1 precede pe P_2 ca în fig. 4.29. Desigur că, şi o relaţie inversă de succedenţă este posibil a se reprezenta, într-o formă asemănătoare cu relaţia de precedenţă.

Pentru interconectarea pieselor de cunoaştere sub aspecte conceptuale şi factuale, apare ca necesară organizarea reţelei semantice în scopul de stabilire a ierarhiei între diferitele componente ale reţelei. O organizare simplă împarte reţeaua în *reţea conceptuală* ce reprezintă concepte de obiecte, acţiuni, procese şi *reţea factuală* ce descrie instanţele acestor concepte.

Relaţia taxonomică între concepte şi instanţe face ca acestea să moştenească proprietăţile conceptelor. Dacă între componentele reţelei factuale apar şi relaţii suplimentare faţă de cele reprezentate în reţeaua conceptuală ce descriu stări de lucruri particulare la care sînt implicaţi numai anumiţi indivizi, ce nu sînt avute în vedere la nivelul conceptelor este necesar să se specifice un mecanism de luare în consideraţie a excepţiilor. Un astfel de exemplu este cel al balenei ce este peste dar naşte pui vii pe care îi alăptează.

4.2.4.3. Reţele semantice extinse

Principala caracteristică a reţelelor semantice extinse este dată de specializarea nodurilor şi arcelor după cerinţele programării logice, definindu-se sorturi pentru noduri (simboluri funcţionale, variabile, constante), sorturi

pentru arce (relaţii condiţionale şi conclusive). Corespondenţa între funcţia semantică şi relaţia folosită se face pe baza oricărei metode pentru reţelele semantice sortate, fiind posibile extensii prin intermediul relaţiilor multisortate ce sînt definite pe baza mai multor criterii. Dacă la o reţea semantică extinsă se mai adaugă facilităţi specifice se obţine metoda de reprezentare de tip hiperreţea. Principalele criterii de diferenţiere a tipurilor de reţele semantice extinse se bazează pe:

- setul primitivelor semantice şi metodele de tratare a neprimitivelor;
- stabilirea sorturilor pentru noduri şi arce;
- mecanismele de definire a altor sorturi (definirea pieselor de metacunoaştere);
- convenţiile de reprezentare grafică.

Reprezentarea prin hiperreţele încearcă să elimine dezavantajele majore ale metodelor tradiţionale legate de problemele specifice de compoziţie, decompoziţie, probleme dominante în sistemele cu caracter tehnic cum sînt proiectarea asistată de calculator, sistemele expert pentru conducerea roboţilor industriali. Ideea de bază a acestei reprezentări porneşte de la asimilarea modului în care este conştientizată cunoaşterea ca urmare a aprofundării. În primul moment se definesc entităţi denumite şi obiecte pentru care se produc piese de cunoaştere, iniţial atomice, deosebite în principal prin nume. Legăturile dintre aceste obiecte determină semnificaţii prin dependentele dintre ele, formînd aspectele relaţionale specifice lumii reale. Odată cu rafinarea cunoaşterii se ajunge la concluzia că obiectele considerate iniţial ca fiind atomice sînt de fapt obiecte compuse ce au structură. Obiectele ce intră în componenţa acestora sînt considerate noi obiecte atomice cu termenii cărora se încearcă descrierea structurii noilor obiecte compuse, componenţa şi legăturile acestora. În final de la o relaţie presupusă binară se obţine un graf de aritate superioară.

4.2.5. Reprezentarea cunoaşterii cu ajutorul cadrelor

Teoria cadrelor este introdusă de Marvin Minsky şi încearcă să reunească o parte din conceptele de bază ale metodelor de reprezentare procedurală, sisteme de producţii şi reţele semantice. Conceptul de cadru este o generalizare a gramaticii de cazuri din lingvistică ce cuprinde informaţia declarativă şi procedurală. Un cadru este reprezentat printr-o structură stereotipă ce utilizează categoriile sintactice:

- *identificator* - are asignat un nume care asociat cu alte descrieri specifică structura cadrului la care este ataşat;

- *forma* - categorii de caracteristici la care se asignează simboluri relaţionale specifice conceptului;
- *faţeta* - sînt reprezentate de perechi *simbol-valuare* cu care se descriu obiectele din relaţiile specificate de formele reprezentării.

Cadrul este format dintr-o structură arborescentă de locuri în care piesa de cunoaştere va plasa simbolurile purtătoare de semnificaţie. Un formalism metalingvistic de reprezentare a cadrelor este:

```
<cadru> =                (<identificator_cadru> <descriere_formă>)
<descriere_formă>=       ((<identificator_formă> <descriere_faţetă>),
                         .....
                         (<identificator_formă> <descriere_faţetă>)).
<descriere_faţetă>=      ((<identificator_faţetă> <descriere_dată>)),
                         .....
                         (<identificator_faţetă> <descriere_dată>)).
<descriere_dată> =       (<valoare>) (<descriere_date> <valoare>)
<valoare> = (<date> <date> <eticheta> <mesaje> <comentariu>).
```

Implementările cunoaşterii utilizînd conceptul de cadru diferă prin detaliile specifice acestora, aşa că anumite implementări orientate spre limbajul LISP dau pentru neterminalele <identificator_cadru>, <identificator_formă>, <identificator_faţetă> reguli de definire diferenţiate prin nivelul de plasare a simbolului în structura de listă. Se foloseşte ca prim simbol identificatorul cadrului, al doilea identificatorul formei, al treilea cel al faţetei, descrierile nestructurate avînd asociate cuvinte cheie pentru cadre, forme, faţete. Simbolurile pot reprezenta proprietăţi ale obiectelor sau relaţiilor, nume de alte cadre, identificatori de proceduri ataşate.

Operaţia de populare în locurile structurii a simbolurilor şi valorilor poartă denumirea de umplere sau "fill-in". Se disting în reprezentarea prin cadre o serie de primitive semantice ce fac să plaseze cunoaşterea în conceptul mai larg oferit de structura cadrului cum sînt:

- forme utilizate la descrierea piesei de cunoaştere;
- forme ce exprimă reprezentări legate de utilizarea cadrului;
- forme corespunzînd prezumpţiilor considerate adevărate;
- forme ce indică reorientarea continuării în situaţia eşecului.

Relaţiile aferente utilizării formelor, relaţii ce caracterizează piesa de cunoaştere şi legăturile sale cu universul problemei se pot împărţi după tip în:

- *generalizare* - specifică concepte cu definiţii mai puţin restrictive;
- *specializare* - specifică concepte cu descrieri ce satisfac o condiţie dată;
- *apartenenţa* - indică clasa din care face parte piesa de cunoaştere şi prin care se moştenesc proprietăţile;

- *compoziţia* - indică obiectele ce intră în alcătuirea obiectului;
- *proprietăţile* - reprezintă proprietăţile specifice ale obiectului reprezentat.

Se dă mai jos reprezentarea cadru pentru o universitate cu profil tehnic care foloseşte forme cadru specifice de descriere

```
cadru: universitate tehnică
       forma: generalizare
       faţeta: institut de învăţămînt superior
       forma: specializare
       faţeta: pregătire tehnică
forma: apartenenţă
       faţeta: universitate
forma: adresa
       faţeta: strada
       faţeta: număr
       faţeta: cod poştal
       faţeta: oraş
       faţeta: ţara
forma: compoziţie
       faţeta: conducere
       valoarea: rector
       valoarea: prorector ştiinţific
       valoarea: prorector administrativ
       valoarea: secretariat ştiinţific
       faţeta: compartimente
       valoarea: electric
       valoarea: mecanic
       valoarea: chimic
forma: proprietăţi
       faţeta: pregătire
       valoare: curs de zi
       valoare: curs seral
       valoare: curs postuniversitar.
```

Piesele de cunoaştere satisfac definiţia cadrelor dată mai înainte. Reglementările privind utilizarea cadrului au în vedere relaţiile acestuia cu descrieri de operaţii, acţiuni, procese. Prelucrările aferente reprezentării prin cadre privesc operaţiile ce se efectuează asupra pieselor de cunoaştere cît şi operaţii ce au ca rezultate obiecte printre care se află piese de cunoaştere.

Această metodă porneşte de la reprezentarea oferită de LISP, la care cunoaşterea este văzută ca perechi atribut_valoare, aspectele relaţionale fiind mai puţin relevante decît la reprezentarea prin reţele semantice. Raţionamentul implică acţiune, motiv pentru care este necesară reprezentarea cunoaşterii despre acţiune. Acţiunea se referă la descrierea acţiunii propriu-

zise formată din cadre ale căror forme reprezintă paşi ai acţiunii în care faţetele sînt caracteristicile paşilor (operanzi, cazuri, obiecte), descrierea de acţiuni implicite prin proceduri la care obiectul va fi plasat în modelul unei situaţii problematice. O procedură ataşată simbolului dintr-un cadru se numeşte *procedură de comutare* dacă simbolul reprezintă un nume de clasă şi procedura este invocată pentru orice element aparţinînd clasei, atunci cînd el substituie în procesul de interpretate simbolul clasei în cadrul considerat. Un exemplu tipic este legat de procedurile de umplere a valorilor asociate formelor într-o reprezentare prin cadre, în care procedura asociată este de forma:

cadru: individ
forma: nume
faţeta: comutaţie
valoare: umple nume (nume).

Procedura asigură umplerea cu simboluri şi valori a formei nume ce reprezintă un cadru referit procedural. Se creează o procedură identică cu apelarea de subrutine din mai multe cadre sau din mai multe puncte ale aceluiaşi cadru. Dacă procedura este ataşată la un simbol unic atunci se încorporează în corpul cadrului imediat următor simbolului de invocare. Alte proceduri folosesc ataşarea implicită şi sînt invocate nu ca urmare a specificării în cadrele unde figurează simboluri la care se ataşează proceduri ci ca urmare a apariţiei unei situaţii specifice. În acest caz procedura se cuplează la apariţia situaţiei, motiv pentru care este numită şi *demon*. Supravegherea situaţiei specifice de către demon se realizează prin:

- apariţia unui fapt în baza de cunoştinţe ce corespunde unui patern;
- apariţia în procesul de interpretare a unei condiţii de invocare pentru un patern dat.

Instalarea demonului în LISP se poate face printr-o funcţie definită de programator avînd o structură de tipul

<demon> = (def_demon<nume><tip><patern><corp>)

4.2.5.1. Reprezentarea acţiunii în cadre

Pentru reprezentarea acţiunilor se poate asocia unei forme simbol aparţinînd categoriei semantice a programelor cu faţete ce reprezintă secvenţe de operaţii, secvenţe de stări sau secvenţe de evenimente. Astfel cadrul conţine pe lîngă componente declarative şi o componentă cu caracter procedural. Conceptul de scenariu face posibilă sistematizarea modului de folosire a substructurilor cadrului în vederea folosirii interpretative a acestuia. Partea procedurală are o structură ce reproduce conceptul de program ce indică

acţiunea normală, acţiunea în cazul lipsei unui obiect, acţiunea în caz de eroare. Partea declarativă, conţine informaţii despre iniţierea şi completarea acţiunii astfel încît să fie anticipate toate aspectele sale funcţionale. Fie de exemplu următorul scenariu:

scenariu: curs
agent: cadru didactic
loc_desfăşurare: sala de curs
timp: 25 aprilie 1991, ora 8
program:
    eveniment_1: intrarea studenţilor în sală
    eveniment_2: intrarea profesorului în sală
    eveniment_3: desfăşurare prima oră de curs
    eveniment_4: pauză
    eveniment_5: intrarea în sală
    eveniment_6: desfăşurarea celei de a doua oră de curs
    eveniment_7: profesorul părăseşte sala
    eveniment_8: studenţii părăsesc sala
început: eveniment_1
obiectiv: eveniment_7.

Acest scenariu poate fi reprezentat ca o succesiune de forme ce conţin simboluri cu pointeri la cadre de detaliu. În exemplul de mai sus profesorul este reprezentat printr-un cadru ce are în compoziţie forme ce precizează entitatea. În procesul de interpretare faptele urmăresc succesiunea de evenimente, ce se desfăşoară după scenariul specificat fără a exista evenimente marcate cu probabilităţi de producere. Se poate stabili că dacă evenimentele 1, 2 şi 5 s-au produs atunci şi evenimentele 3 şi 6 se vor produce cu siguranţă. Renunţînd la imaginea de perfecţiune se poate determina un program de evenimente ce specifică alternativele posibile în caz de eşec. Demonii instalaţi au rolul de a trata cazuri cum este cel profesor bolnav, şi va indica trecerea la evenimentul_8.

4.2.5.2. Introducerea regulilor în cadre

O facilitate puternică este cea a introducerii în cadre a reprezentării regulilor. Transpunerea unei reguli în cadre porneşte de la obiectivele:

- regula conţine două acţiuni distincte deci două forme procedurale diferite;
- prima formă cea de evaluare a condiţiilor are ca efect autorizarea execuţiei celei de a doua forme procedurale sau ieşirea din regulă la eşec;

- a doua formă determină corpul propriu-zis al acţiunii ce se execută cînd condiţia este îndeplinită;
- contextul operaţional este reprezentat de celelalte forme cu caracter declarativ.

Se consideră în exemplul de mai jos regula:

```
IF: rezultatul testului de citire-scriere este eronat, bitul 2
blocat în 1
   (and) se manifestă la circuitul M1
   (and) circuitul M2 funcţionează corect
THEN: circuitul M1 este defect
   (and) reparare prin înlocuire
```

se va reprezenta prin cadre astfel:

Cadru R1
  forma: condiţie
    faţeta: rezultat test citire-scriere eronat, bit 2 blocat în 1
    faţeta: se manifestă la circuitul M1
    faţeta: M2 funcţionează corect
  forma: acţiune
    faţeta: defect
      valoare: M1
    faţeta: reparaţie
      valoare: înlocuire

Structura implicit ierarhizată a conceptului de cadru face posibilă organizarea sub formă de arbori de clasificare. Structura arborescentă derivă din cadre prototip ale claselor şi subclaselor.

5

SISTEME REZOLUTIVE

Un sistem rezolutiv este format din totalitatea componentelor unui sistem de inteligenţă artificială avînd ca obiectiv rezolvarea de probleme. Prin problemă se înţelege o noţiune fundamentală de psihologie a gîndirii ce se referă la dificultatea de natură cognitivă ce se constituie ca moment iniţial al activităţilor inteligente. Dificultatea porneşte de la imposibilitatea asocierii unor răspunsuri la întrebările asociate unei entităţi necunoscute. Problema formulată spre rezolvare unui sistem de inteligenţă artificială este un enunţ de problemă potenţială, confirmarea enunţului unei astfel de probleme externe, reale, depinde de apariţia unei situaţii discordante între obiectivele problemei, faptele din baza de cunoştinţe, metodele disponibile pentru corelarea cu obiectivele problemei. Transpunerea problemei de la subiectul uman la sistmul de inteligenţă artificială presupune reformularea acesteia într-un formalism ce reflectă situaţia problematică actuală creată ca urmare a discordanţei între obiectivele specificate în formularea externă şi resursele cognitive şi rezolutive ale sistemului.

Mecanismul de inferenţă este inima oricărui sistem expert, care alimentat de baza de cunoştinţe construieşte raţionamentul în mod dinamic decizînd ce reguli sînt declanşate şi în ce ordine. Astfel, raţionamentele declanşate pentru reprezentarea prin reguli de producţie sînt cele în logica formală *"modus ponens"* şi *"modus tollens"*.

Componentele primitive constituente ale problemei pot fi grupate în:

- aserţiuni referitoare la obiecte abstracte sau concrete formînd pentru problema dată universul discursului. Acestea alcătuiesc aspectul factual al enunţului cunoscut sub denumirea de date ale problemei;
- aserţiuni explicite sau implicite referitoare la operaţiile permise a fi aplicate obiectelor menţionate. Aceste aserţiuni formează aspectul procedural al enunţului, cunoscut şi sub denumirea de condiţii ale problemei;

- obiecte, proprietăţi a căror existenţă este o consecinţă a enunţului formînd ceea ce se numesc "necunoscute ale problemei".

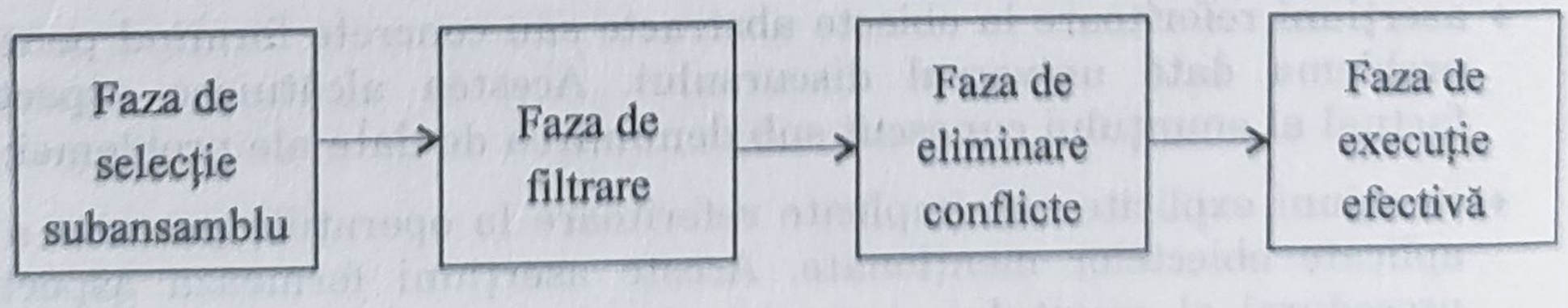
Obiectivul rezolvării oricărei probleme este de a furniza o determinare pentru fiecare din necunoscute. Universul discursului în procesul rezolvării este delimitat mai întîi de datele şi condiţiile formulate prin enunţ. La sistemele inteligente bazate pe cunoaşterea universului iniţial i se adaugă piese de cunoaştere din baza de cunoştinţe ce sînt selectate fie prin referire directă la enunţ fie prin inferente logice cu premise formulate în enunţ.

5.1. Ciclul de bază al unui mecanism de inferenţă

Indiferent de modul de raţionare utilizat ciclul de bază al unui mecanism de inferenţă cuprinde patru faze (fig. 5.1) şi anume:

- faza de *selecţie* a unui subansamblu al bazei de fapte şi reguli ce merită atenţia faţă de restul bazei. Selecţia oferă o economie de timp considerabilă pentru fazele următoare. Desigur că sistemul de inferenţă nu trebuie să privilegieze un grup de reguli faţă de altele;
- faza de *filtrare* ce are ca efect comparaţia între partea de premisă a regulii considerate şi faptele bazei de fapte pentru determinarea regulilor aplicabile. Aplicarea după faza de selecţie a fazei de filtare ajută la reducerea substanţială a ansamblului regulilor ce sînt filtrate cu elementele din baza de fapte;
- faza de *rezolvare a conflictelor* are ca obiectiv alegerea acelor reguli ce sînt aplicate efectiv. Problema este rezolvată printr-o strategie ce poate fi foarte simplă în raport cu contextul sau mai complexă ţinînd cont de context în sensul aplicării regulii cît mai promiţătoare;
- faza de *execuţie* constă din aplicarea regulilor alese mai înainte, acţiunea constînd în general în adăugarea de noi fapte în baza de fapte. Este posibil ca aplicarea regulii să facă apel la proceduri externe avînd ca efect modificări ale bazei de fapte, formularea de întrebări către utilizator.

▼ ***Fig. 5.1. Fazele procesului rezolutiv***



Lanţurile inferenţiale sau reţelele inferenţiale între premise şi concluzie determină procesul numit şi *raţionament,* ce se bazează pe faptele furnizate prin enunţ şi pe piesele de cunoaştere existente în baza de cunoştinţe. Atunci cînd faptele furnizate în enunţ şi piesele de cunoaştere din baza de cunoştinţe sînt suficiente, raţionamentul se numeşte *direct,* întrucît entităţile din reţeaua inferenţială aparţin faptelor aferente problemei. Incompletitudinea faptică duce la impas, şi au fost studiate şi alte tipuri de raţionamente cum este de exemplu raţionamentul *metaforic* sau raţionamentul prin analogie.

Metodele binecunoscute pînă în prezent sînt cele ce fac parte din clasa raţionamentelor directe. Metodele ce studiază transformările de la premise la concluzii se numesc şi metode de raţionare formală şi au o importanţă deosebită în abstractizarea unor proprietăţi ce intervin la raţionamentele prin analogie.

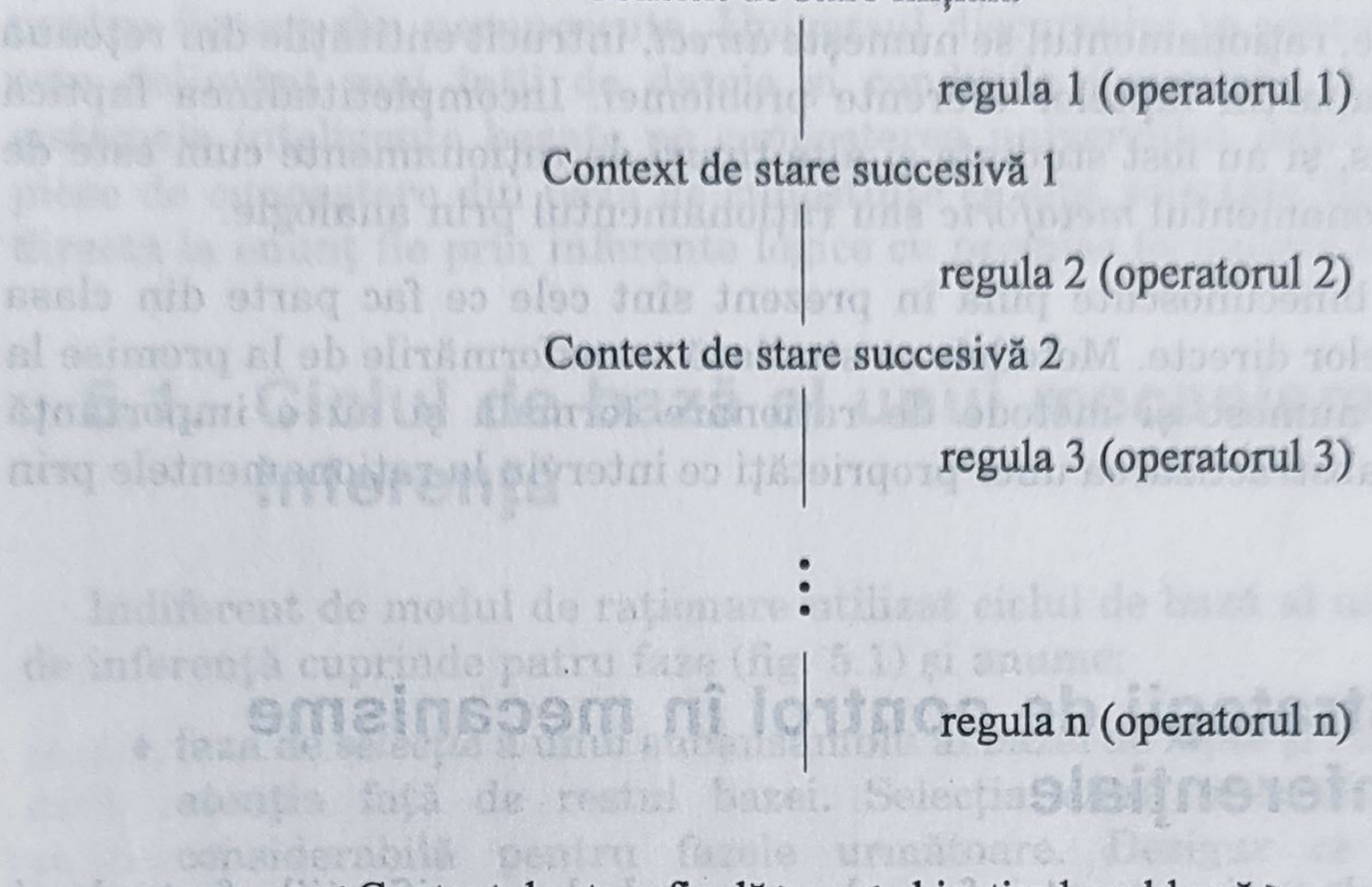
5.2. Strategii de control în mecanisme inferenţiale

Metodele de raţionament informal pornesc de la specificaţiile factuale şi procedurale cuprinse în enunţ. Aceste metode de raţionament surprind interpretări structurale, semnificaţii de natura relaţională, proprietăţi primitive, structuri ierarhice bazate pe primitive semantice. Raţionamentul informal este necesar să se valideze prin aplicarea regulilor specifice simbolurilor folosite actual. Desfăşurarea raţionamentelor are loc pe baza unei strategii ce asigură succesiunea inferenţelor. Se dau mai jos cîteva strategii de control al raţionamentelor.

5.2.1. Strategia de control înainte

Această strategie porneşte de la starea iniţială de fapte descrisă prin enunţ, stare de la care prin aplicarea de raţionamente se generează candidaţi la soluţie pînă la obţinerea răspunsului corespunzător obiectivului problemei. Regulile ce se utilizează pot fi reguli de inferenţă ale mecanismelor logice de bază, reguli specifice definite prin enunţ referitoare la generarea termenilor, simboluri relaţionale, restricţii ce reduc spaţiul problemei. Operatorii corespunzători algebrei multisortate ce defineşte problema pot fi examinaţi prin reguli. Se mai spune în strategiile de control înainte că se porneşte de la fapte pentru a ajunge la obiectiv. În consecinţă se vor selecţiona regulile a căror parte de condiţie este verificată (faza de selecţie şi filtrare). Pentru faza de rezolvare a conflictelor din ansamblul regulilor selectate se vor alege acelea ce au prioritate maximă. Ca urmare, se va obţine o strategie generală de control similară cu cea prezentată în fig. 5.2.

▼ ***Fig. 5.2. Strategia de control înainte***



Aplicarea regulilor determină date şi fapte noi. Se pot obţine astfel la fiecare pas derivări valabile rezultate ca urmare a aplicării mai multor reguli, dar numai una este valabilă în sensul că aparţine lanţului ce conduce la soluţie. La acest moment nu se cunoaşte această regulă şi ca urmare se vor aplica toate regulile ce sînt posibil aplicabile obţinînd o mulţime de stări caracterizate prin date şi fapte. Succesiunea sugerează un arbore ale cărui noduri sînt stări în spaţiul stărilor, soluţia problemei fiind dată printr-o procedură de căutare în spaţiul stărilor. Ţinînd cont de caracteristicile metodei ce se bazează pe strategia de control înainte, metoda mai este numită şi productivă, iar strategia se mai numeşte şi *strategie de căutare dirijată prin date sau de jos în sus*. Procesul de căutare continuă pînă cînd mulţimea regulilor aplicabile devine vidă. Un astfel de proces poate fi transpus în pseudocod astfel:

```
Stabilirea faptelor inițiale
begin
  faza de filtrare (determinarea mulțimii regulilor aplicabile)
  while (ansamblul regulilor aplicabile nu este vid și problema nu
         a fost rezolvată)
    execută faza de lucru (rezolvarea conflictelor)
         aplicarea regulii alese
         modificarea ansamblului regulilor aplicabile
    end executa
  end while
end
```

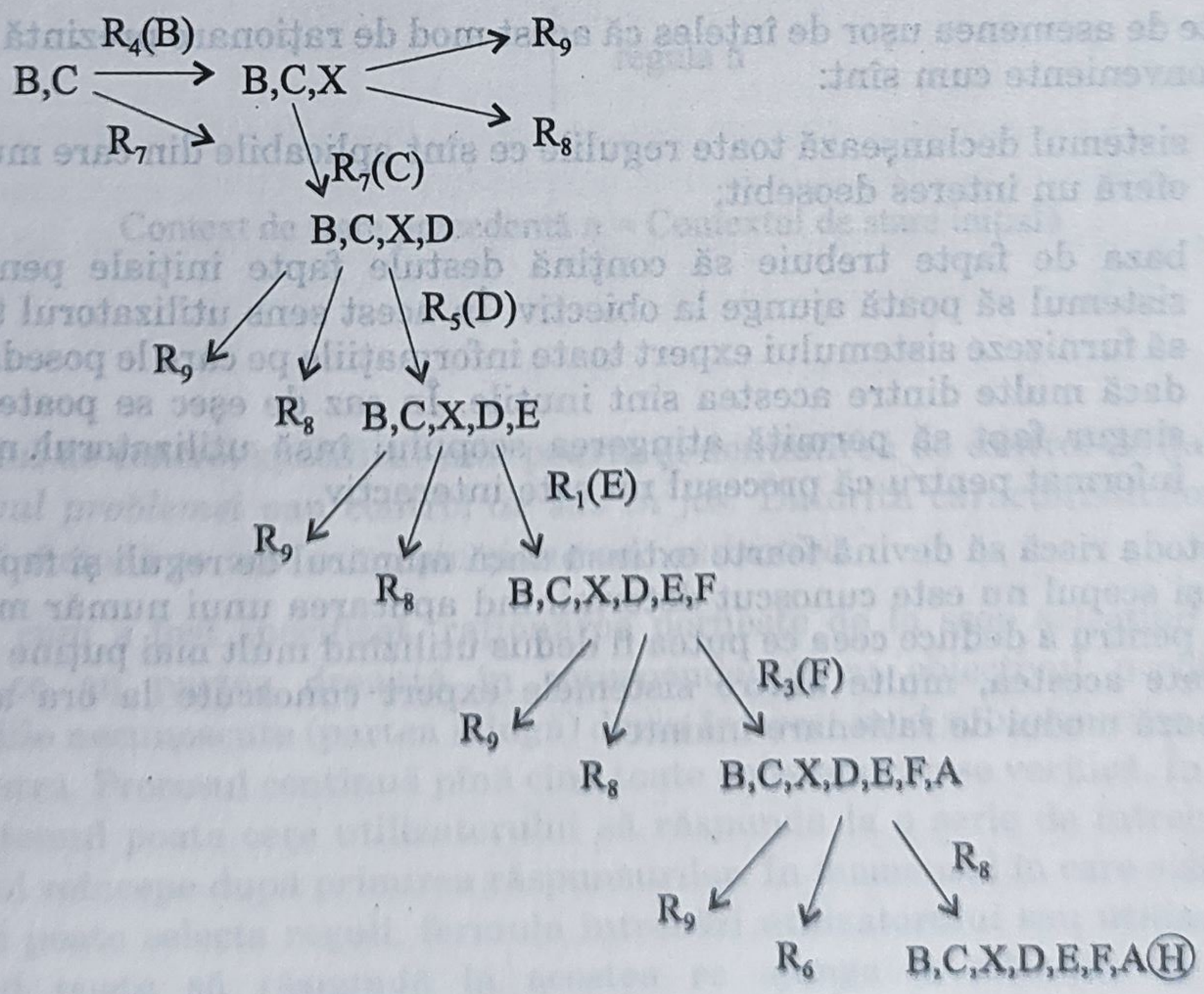
Se poate spune că eficacitatea mecanismului de inferenţă constă în pertinenţa deciziei luate în faza de lucru, fapt ce determină şi rapiditatea cu care sistemul ajunge la soluţie. Ca efect, rezolvarea unei probleme poate determina un număr foarte mare de cicli de bază. Este cert că pentru un calculator universal fiecare ciclu de bază determină un număr mare de cicluri instrucţiune. Viteza de execuţie este de ordinul miilor de ciclii de inferenţa pe secundă la milioane de ciclii de infereţă (LIPS Logical Inference Per Second). Pornind de la această observaţie este justificat interesul de scădere pe cît posibil a numărului ciclilor de inferenţă.

Se consideră mai jos o bază de cunoştinţe reprezentată prin reguli de producţie:

R_1: IF B and D and E THEN F
R_2: IF D and G THEN A
R_3: IF C and F THEN A
R_4: IF B THEN X
R_5: IF D THEN E
R_6: IF A and X THEN H
R_7: IF C THEN D
R_8: IF X and C THEN A
R_9: IF X and B THEN D

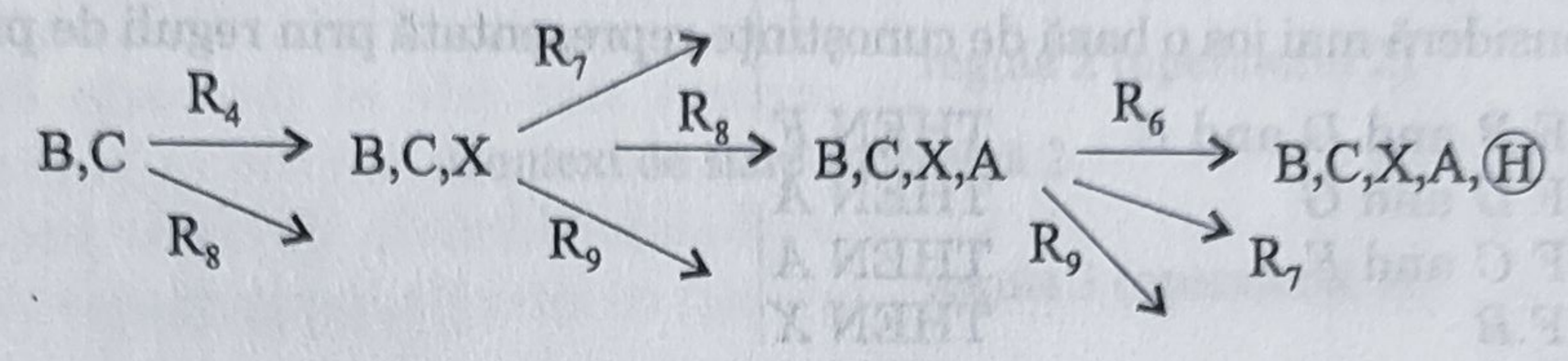
Dacă se presupune baza de fapte iniţială B, C şi obiectivul H, construcţia arborelui de decizie pornind de la prima regulă prin secvenţierea regulilor în ordinea în care apar în baza de cunoştinţe se obţine următorul arbore (fig. 5.3):

▼ ***Fig. 5.3. Aplicarea regulilor în ordinea stocării***



Se consideră la acest moment o strategie prin care se selectează din mulţimea regulilor aplicabile acelea pentru care numărul de condiţii din partea de premiză este mai mare. Se observă că regula R_8 este preferabilă faţă de regula R_7 întrucît are în premisă două condiţii faţă de R_7 care are doar una. Cu această strategie se obţine arborele din fig. 5.4.

▼ ***Fig. 5.4. Selecţia regulilor funcţie de numărul condiţiilor din premisă***



Se observă că printr-o astfel de strategie numărul de inferenţe se reduce la trei faţă de şase în cazul anterior. În plus trebuie reţinut faptul că o inferenţă aplicată odată nu se mai aplică întrucît nu serveşte la nimic să se deducă un fapt ce a fost deja dedus. Eficacitatea ajungerii la scop este dată de numărul de inferenţe necesare, număr de inferenţe ce depinde în mare măsură de faza de eliminare conflicte. Strategia utilizată în fig. 5.3 deşi este foarte simplă are dezavantajul de a fi puternic dependentă de ordinea regulilor din baza de cunoştinţe. Chiar dacă strategia utilizată în fig. 5.4 este mai complicată din punctul de vedere al implementării se justifică prin rezultatele obţinute, mai ales prin rapiditate.

Este de asemenea uşor de înţeles că acest mod de raţionare prezintă o serie de inconveniente cum sînt:

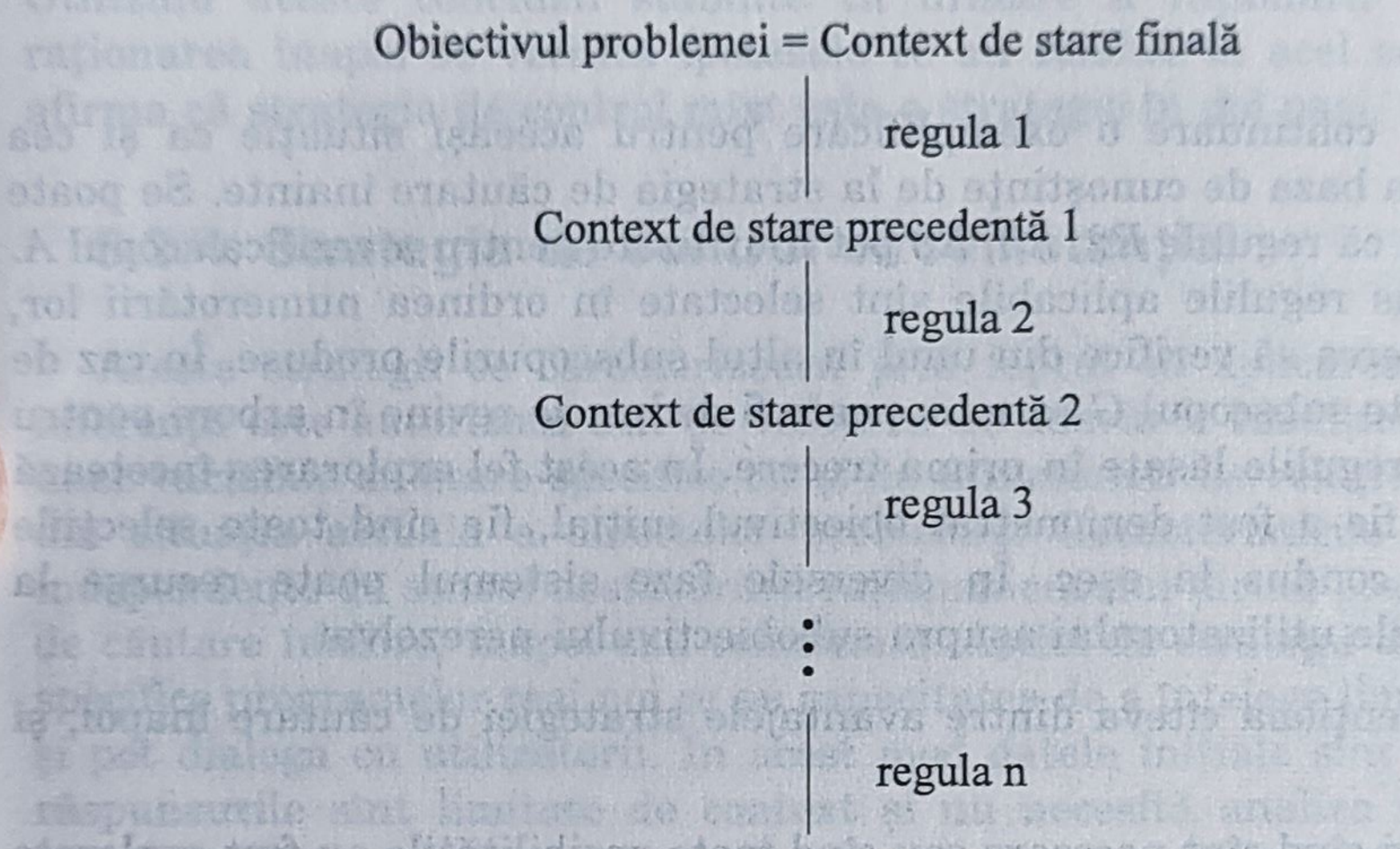
- sistemul declanşează toate regulile ce sînt aplicabile din care multe nu oferă un interes deosebit;
- baza de fapte trebuie să conţină destule fapte iniţiale pentru ca sistemul să poată ajunge la obiectiv. În acest sens utilizatorul trebuie să furnizeze sistemului expert toate informaţiile pe care le posedă chiar dacă multe dintre acestea sînt inutile. În caz de eşec se poate ca un singur fapt să permită atingerea scopului însă utilizatorul nu este informat pentru că procesul nu este interactiv.

Metoda riscă să devină foarte extinsă dacă numărul de reguli şi fapte este mare şi scopul nu este cunoscut determinînd aplicarea unui număr mare de reguli pentru a deduce ceea ce putea fi dedus utilizînd mult mai puţine reguli. Cu toate acestea, multe dintre sistemele expert cunoscute la ora actuală utilizează modul de rationare înainte.

5.2.2. Strategia de control înapoi

Această strategie de control porneşte de la obiectivul problemei care prin aplicarea regulilor de descompunere se transformă în subprobleme de complexitate mai mică. Structura generală a inferenţierii utilizînd metoda de control înapoi este ilustrată în fig. 5.5.

▼ *Fig. 5.5. Strategia de raţionare înapoi*



Modul de control specificat mai poartă şi denumirea de control dirijat prin *obiectivul problemei* sau control *de sus în jos*. Datorită caracteristicilor sale această metodă se mai numeşte şi *metodă reductivă*.

Aşa cum a fost specificat, raţionarea porneşte de la scop selectînd acele reguli ce au partea dreaptă în corespondenţă cu obiectivul problemei. Condiţiile necunoscute (partea stîngă) devin în acest mod subscopuri ce se vor demonstra. Procesul continuă pînă cînd toate subscopurile se verifică. În acest caz sistemul poate cere utilizatorului să răspundă la o serie de întrebări şi procesul reîncepe după primirea răspunsurilor. În momentul în care sistemul nu mai poate selecta reguli, formula întrebări utilizatorului sau utilizatorul nu mai poate să răspundă la acestea se ajunge la situaţia de eşec. Transpunerea algoritmului în pseudocod este ilustrată mai jos.

```
begin
  faza de filtrare
  IF ansamblul regulilor selectate este vid
  THEN întrebări adresate utilizatorului
  ELSE
   while problema nu este rezolvată şi mai sînt reguli
   selecţionate
      executa
          faza de lucru
          adaugă subscopul la partea dreaptă a regulii
          selecţionate
          IF subscopul nu este soluţionat reluare
          ELSE editează soluţia şi STOP
      end while
  end if
end
```

Se dă în continuare o exemplificare pentru aceeaşi situaţie ca şi cea prezentată la baza de cunoştinţe de la strategia de căutare înainte. Se poate uşor observa că regulile *R*2, *R*3, *R*8 pot fi utilizate pentru a verifica scopul *A*. În acest sens regulile aplicabile sînt selectate în ordinea numerotării lor, pentru a încerca să verifice din unul în altul subscopurile produse. În caz de eşec, cum este subscopul *G* care nu poate fi dedus, se revine în arbore pentru a se selecta regulile lăsate în prima trecere. În acest fel explorarea încetează atunci cînd fie a fost demonstrat obiectivul iniţial, fie cînd toate selecţiile posibile au condus la eşec. În diversele faze sistemul poate recurge la chestionări ale utilizatorului asupra subobiectivului nerezolvat.

Se pot menţiona cîteva dintre avantajele strategiei de căutare înapoi, şi anume:

- atunci cînd sînt necesare sau cînd toate posibilităţile au fost explorate sistemul recurge la întrebări adresate utilizatorului;
- arborele de căutare este adesea mai puţin adînc decît cel aferent strategiei de căutare înainte;
- procesul de raţionare este interactiv.

Pe lîngă avantajele menţionate mai sus strategia de control înapoi prezintă şi un dezavantaj major legat de faptul că există pericolul buclării, în sensul că pentru a demonstra *A* trebuie demonstrat *B* iar pentru demonstrarea lui *B* trebuie demonstrat *A*.

5.2.3. Strategia de control combinat

Aceste strategii de control sînt caracterizate de faptul că utilizează metode reductive pentru descompunerea problemei în subprobleme care apoi sînt rezolvate prin metode productive. Rezultatul aplicării parţiale a unei metode reductive este descompunerea acesteia în subprobleme ce pot fi tratate în continuare fie prin aplicarea metodelor reductive fie prin schimbarea metodei de tratare şi aplicarea, de exemplu, a metodelor productive. Succesiunea

inversă nu este posibilă întrucît metodele productive generează de la primul pas candidaţi la soluţie ce nu reprezintă subprobleme fapt pentru care odată selectată metoda productivă nu poate fi părăsită pentru a continua rezolvarea prin alte metode. Deci odată generată o metodă productivă aceasta va fi utilizată pînă la obţinerea soluţiei finale.

Utilizarea combinată a celor două metode duce la o simplificare a problemei şi are ca efect o reducere a spaţiului stărilor pentru subproblemele componente cu efecte favorabile în privinţa necesarului de resurse pentru rezolvare. Cu aceste considerente este clar faptul că nu trebuie confundată strategia de inferenţă mixtă, înainte în alternanţa cu strategia de control înapoi cu metoda de raţionare mixtă. În general un sistem expert utilizează strategia de control înainte pentru determinarea unor concluzii parţiale. Utilizînd aceste concluzii stabilite ca urmare a raţionării înainte, prin raţionarea înapoi se verifică ipotezele ce au condus la acel scop. Se poate afirma că strategia de control mixt este o strategie în doi paşi.

5.2.4. Strategia de control circumstanţial

Aceste strategii se caracterizează prin faptul că aplicarea regulilor de inferenţă este autorizată atît de valoarea de adevăr a condiţiei, de termenii unor variabile de stare specifice cît şi de caracteristicile relaţionale derivate din situaţia actuală a obiectelor problemei. Caracteristicile metodei sînt independente de sensul desfăşurării raţionamentului făcînd posibile strategii de căutare înainte, înapoi sau combinate. Astfel de strategii de control sînt specifice programelor mai noi ce au capacitatea de a înţelege limbajul natural şi pot dialoga cu utilizatorii. În acest mod datele iniţiale sînt mult reduse, răspunsurile sînt limitate de context şi nu necesită analiza răspunsurilor complexe. Un mod simplu de realizare se bazează pe întrebările puse atunci cînd mecanismul de inferenţă le foloseşte şi nu pe o cantitate mare de informaţii standard la începutul consultării. În funcţie de caracteristicile problemei între utilizator şi sistem are loc un dialog inteligent bazat pe "reflexiile sistemului".

5.2.5. Alegerea strategiei de control

În funcţie de tipul de raţionament ales, construcţia mecanismului de inferenţă necesită două etape delicate. Prima se referă la definirea metodelor de selecţie a regulilor candidate. În acest sens se confruntă fiecare regulă cu ansamblul faptelor din baza de fapte sau cu scopul ce se doreşte atins. În alţi termeni operaţia se mai numeşte şi filtrare. A doua etapă se referă la alegerea regulii pentru declanşare. Această alegere influenţează performanţele sistemului. Dacă un om ştie ce cunoştinţe utilizează într-o situaţie dată un sistem expert va trebui să urmeze o strategie. Un mod de control foarte interesant se bazează pe utilizarea metacunoştinţelor. Acest nivel de cunoaştere este specific strategiilor de raţionare. Cert este faptul că nici o

strategie de control nu este bună în orice situaţie. Controlul dirijat prin date este foarte popular, uşor de realizat şi se recomandă a se utiliza atunci cînd se dispune de o cantitate mare de informaţie neştiind nimic despre scopul ce urmează a fi atins. Strategia de control înainte este indicată pentru tratarea cunoştinţelor empirice şi este ineficientă pentru probleme complexe. Dacă însă unul sau mai multe scopuri trebuiesc atinse sau verificate este mult mai raţional să se utilizeze strategia de control înapoi. Raţionarea înapoi este recomandată în cazul informaţiilor incomplete sau cînd se poate angaja un dialog cu utilizatorul.

5.3. Analiza tipurilor de probleme

Capacitatea de rezolvare a problemelor de către un sistem inteligent este apreciată după uşurinţa cu care dă soluţii la probleme ce nu au fost stabilite în prealabil, precizarea lor fiind făcută în momentul încărcării bazei de cunoştinţe.

Din punctul de vedere al satisfacerii condiţiilor de formulare se pot distinge trei mari categorii de probleme:

- *probleme bine formulate* - sînt acele probleme ce satisfac condiţiile de necesitate şi suficienţă pentru toate componentele din structură, necunoscutele împreună cu datele alcătuind un model consistent;
- *probleme incomplet definite* - sînt problemele în a căror formulare se poate pune în evidenţă lipsa unor date, a condiţiilor acestora sau sînt lipsuri în specificarea elementelor văzute ca necunoscute;
- *probleme greşit formulate* - pentru care pot fi puse în evidenţă contradicţii şi inconsistente sau nu pot fi stabilite clar componentele problemei.

Aplicarea strategiilor de rezolvare specifice tipului de problemă se face după o analiză asupra naturii intime a acesteia, în vederea clarificării de detaliu. Problemele bine formulate pot fi clasificate în: probleme de tip interogativ, probleme de tip predicativ, probleme de tip inductiv.

O problemă de tip interogativ este compusă din trei părţi:

- *ipoteza* (alcătuită din date);
- *procesul* la care sînt supuse datele;
- *rezultatul* ce se obţine în urma aplicării procesului.

O formalizare completă a problemei de tip interogativ presupune definirea completă a două din cele trei elemente, cea de a treia fiind necunoscută. Dacă se notează cu I ipoteza, cu P procesul, cu R rezultatul şi cu X necunoscuta,

funcţie de poziţia necunoscutei în componentele problemei se deosebesc probleme interogative de tip *IPX, IXR* şi *XPR.* În situaţia în care toate componentele menţionate sînt cunoscute enunţul problemei este de tip *IPR* şi aceasta reprezintă un fapt. Se poate afirma că atunci cînd unul din componentele unui fapt este necunoscut acesta devine o problemă.

O analiză de detaliu arată că problemele interogative de tip *IPX* au enunţul "*care este rezultatul x al aplicării procesului P asupra obiectelor din ipoteza I?*" şi sînt de fapt problemele clasice de prelucrare a datelor. Desigur că enunţul problemei este cunoscut sau completat cu informaţii preluate din baza de cunoştinţe. Problema interogativă de tip *IXR* corespunde enunţului "*ce proces X trebuie aplicat obiectelor din ipoteza I pentru a obţine rezultatul R?*" este una din problemele delicate ce are un caracter creator, la a cărei rezolvare vor trebui luate în consideraţie următoarele abordări:

a) rezolvare pe baza unor reguli euristice de tipul
   IF (ipoteza este de tipul (I))
   and (rezultatul este de tip (R))
   THEN se aplică (metoda 1) sau (metoda 2)... sau (metoda *n*)
b) rezolvare pe baza raţionamentelor prin analogie.
c) rezolvare pe baza metodelor de sinteză procedurală.

Problema interogativă de tip *XPR* corespunde formulării "*ce obiecte X trebuiesc prelucrate pentru a obţine rezultatul R?*". Problema poate fi rezolvată direct doar dacă P este o bijecţie adică ştiind că $R = P(X)$ soluţia va fi $X = P^{-1}(R)$. Dacă însă P nu este o bijecţie atunci abordarea trebuie făcută prin metode reductive.

Problemele de tip predicativ au structura similară, componentele lor fiind:

- *ipoteza,* (*I*);
- *procesul inferenţial,* (*P*);
- *concluzia,* (*C*).

O astfel de problemă este notată *IPC.* Chiar dacă structural problemele de tip predicativ şi cele de tip interogativ sînt asemănătoare, diferă prin faptul că la problemele de tip predicativ ipoteza şi concluzia sînt entităţi cu valoare de adevăr. O problemă predicativă în care concluzia este o necunoscută se numeşte *derivare.* Derivarea apare ca o problemă rar întîlnită în sisteme inteligente dar se întîlneşte frecvent în aplicarea procedurilor ce vizează descompunerea problemelor în subprobleme.

Problema de tip predicativ la care procesul inferenţial este necunoscut se numeşte *demonstraţie.* Soluţia unei astfel de probleme este formată dintr-un lanţ de inferenţe ce porneşte de la ipoteze şi ajunge la concluzie. Aplicarea unui lanţ inferenţial este caracteristică sistemelor rezolutive pentru demonstrarea automată a teoremelor.

Problema de tip predicativ în care ipoteza este cunoscută se numeşte "*inducţie*". Soluţia presupune descoperirea unor concepte iniţiale, a cauzalităţii sau a altor premise prin interpretarea rezultatelor ce se obţin pe baza unor

reguli de inferenţă cunoscute. Procesul rezolutiv ce implementează metode inductive se mai numeşte şi proces de *învăţare* din exemple.

Procesele de tip *imperativ* sînt cele care nu conţin necunoscute în triada ipoteza, proces, rezultat. O problemă de tip imperativ nu necesită date de intrare şi rezultatul procesului este aprioric cunoscut. Rezultatul invocării procesului este efectul actual, adică producerea rezultatului cunoscut. Faptul că o problemă este stratificată pe mai multe nivele conceptuale, face ca o problemă la un anumit nivel să se descompună în probleme de alte tipuri pe nivelele inferioare.

Problemele *incomplet formulate* au soluţie doar dacă în urma aplicării unor procedee ce nu alterează specificul problemei din enunţ, se obţine o nouă formulare ce satisface condiţia de bună formulare. Principalele procedee ce se aplică în aceste cazuri sînt:

- reducerea enunţului la o formă de problemă bine formulată ce este făcută prin eliminarea unor elemente lipsite de relevanţă, elemente se alterau buna formulare;
- completarea enunţului de date transmise prin moştenire, aferente pieselor din enunţ, cu date asumate prin analogie, cu noile date obţinute în urma reformulării de către utilizatorul ce interacţionează cu sistemul rezolutiv;
- descompunerea problemei în subprobleme astfel încît anumite componente să fie bine formulate. Metoda se aplică în general la problemele de complexitate mai mare. Se va obţine astfel o rezolvare parţială în situaţia în care subproblemele complet formulate sînt rezolvate prin procedurile deja descrise;
- extinderea obiectivelor problemei asupra clasei ce cuprinde problema iniţială, introducerea de obiecte abstracte, a condiţiilor suplimentare ce permit reformularea enunţului pentru a se satisface condiţiile de problemă bine formulată.

Este de la sine înţeles faptul că nu se pune problema rezolvării problemelor incomplet formulate. Pot în schimb exista în anumite cazuri analizoare de problema care să ilustreze inconsistenţele de formulare şi să justifice refuzul de a rezolva o anumită problemă de către sistem.

5.4. Procesul rezolvării şi cunoaşterea

Cunoaşterea pentru rezolvarea problemei provine din mecanismele inferenţiale ale sistemului, din enunţul problemei şi din cunoaşterea apriori memorată în baza de cunoştinţe. Clasificarea cunoaşterii despre problemă este o preocupare recentă în inteligenţa artificială avînd ca scop introducerea unei ordini conceptuale în raportul cunoaştere/rezolvare. Se poate organiza cunoaşterea despre problemă folosind concepte ale demonstrării automate a

teoremelor, utilizînd conexiunile între matricile formelor prenex. Bibel propune o ierarhizare pe cinci nivele ce au ca bază: cunoaşterea factuală, cunoaşterea deductivă, cunoaşterea raţionamentală, cunoaşterea despre structurile globale, cunoaşterea despre controlul asupra raţionamentelor. Concepută iniţial pentru metoda conexiunilor între matrici, clasificarea poate fi extinsă la cazul general fără a ţine cont în vreun fel de metoda de rezolvare.

Pornind de la ierarhizarea cunoaşterii se obţine un arbore de clasificare astfel:

- cunoaştere factuală
  - conceptuală
  - instanţială
- cunoaştere procedurală
  - transformaţională
  - inferenţială
- cunoaştere de control
  - pentru structuri
  - pentru raţionamente

Dependenţele reciproce dintre diferitele clase ce formează piesele de cunoaştere pornesc de la cunoaşterea factuală ce defineşte prototipuri ale unor fapte şi instante ce descriu fapte individuale. Asupra pieselor de cunoaştere factuală operează cunoaşterea procedurală ce specifică proceduri de transformare a pieselor de cunoaştere factuală. Cunoaşterea de control asupra procesului de rezolvare se bazează pe piese de cunoaştere procedurală şi specifică modul de succesiune în care se înlănţuie procedurile de transformare pentru obţinerea de noi structuri, modul de succesiune în aplicarea regulilor de inferenţă pentru raţionamente. Se poate considera că la fiecare nivel pot exista:

- piese ale cunoaşterii directe;
- piese ale cunoaşterii deduse în urma aplicării regulilor de inferenţă, ce sînt diferite de piesele cunoaşterii directe. Dacă în baza de cunoştinţe există o parte din aceste piese, o serie de procese inferenţiale, mari consumatoare de timp, nu mai sînt necesare. Luarea deciziei de introducere în baza de cunoştinţe a pieselor de cunoaştere deduse se face după o atentă evaluare a volumului ocupat de acestea în baza de cunoştinţe.

În particular, în sisteme expert apare necesitatea de organizare a cunoaşterii astfel încît să se ţină seama de opiniile mai multor agenţi cunoscători, făcînd în acelaşi timp posibilă separarea cunoaşterii globale comună subiecţilor cunoscători, de cunoaşterea personalizată.

Se tratează mai departe moduri de prezentare a problemelor după diferite formalisme specifice modurilor de reprezentare a cunoştinţelor.

5.5. Reprezentarea problemelor în limbajul calculului cu predicate de ordinul întîi

Abordarea problemelor de către sistemul rezolutiv devine posibilă atunci cînd acestea sînt formulate în acel limbaj de reprezentare ce este înţeles de sistemul ce le va reprezenta. Este deci uşor de înţeles că rezolvarea problemelor exprimate în limbajul calculului de ordinul întîi este adecvată atunci cînd piesele de cunoaştere sînt reprezentate în acest limbaj. Enunţul problemei este în general exprimat într-un limbaj apropiat de cel natural, ce este transcris în formule ale calculului cu predicate de ordinul întîi.

5.5.1. Particularităţi ale reprezentării problemelor în limbajul calculului cu predicate de ordinul întîi

Transcrierea şi formularea presupune o analiză ce are obiectivele:

- descompunerea enunţului problemei în piese de cunoaştere ce pot fi descrise cu ajutorul predicatelor de ordinul întîi;
- extragerea caracteristicilor structurale ale teoriei în care se plasează problema avînd ca scop recunoaşterea tipului de similaritate identificatoare a competenţei sistemului rezolutiv;
- definirea cunoaşterii factuale prin asignarea de simboluri constante la variabilele predicatelor;
- definirea obiectivului problemei cu ajutorul formulelor calculului cu predicate de ordinul întîi.

Caracteristicile structurale ale teoriei în care se plasează problema sînt funcţie de modalităţile de specificare a componentelor problemei în contextul grupării pieselor de cunoaştere specificate în enunţ, de mulţimea suport a obiectelor structurii, de mulţimea simbolurilor funcţionale, relaţionale şi a simbolurilor distincte. Mulţimea suport a structurii cuprinde atît obiectele de tip constantă, organizate în colecţii definite fie enumerativ, fie generativ pe baza descrierii unui prototip obiectual şi al unui mecanism de generare. Simbolurile funcţionale au în general rolul de a furniza termenii ce candidează a fi ataşaţi ca variabile formale sau ca instante, de a furniza termenii posibili ai unor simboluri relaţionale în formule. Simbolurile relaţionale apar în formulele ce descriu enunţul problemei în limbajul calculului cu predicate de ordinul întîi. Aritatea şi ordinul se determină după stabilirea relevanţei semantice.

Eficienţa sistemului rezolutiv poate fi mărită prin eventuala rejecţie a problemelor pentru rezolvarea cărora sistemul nu are competeţă, înainte de declanşarea raţionamentului pentru care tot nu s-ar fi obţinut un succes. În

urma transcrierii problemei în formulele calculului cu predicate de ordinul întîi problema se prezintă sistemului în următoarea grupare de simboluri:

- *suportul obiectual* ce ataşează piese de cunoaştere ca variabile formale pentru expresii predicative primare;
- *simboluri funcţionale* formate din piese de cunoaştere pentru definirea termenilor din formule;
- *simboluri predicative* ce definesc piese cu valoare de adevăr ce descriu proprietăţi ale obiectelor sau relaţii între obiecte;
- *fapte* - expresii formale ce exprimă adevăruri despre obiectele implicate, cuprinzînd reguli specifice teoriei de reguli ale problemei. După unii autori totalitatea formulelor formează *axiomele problemei;*
- *obiectivul* reprezentat printr-o expresie formală, prin care se exprimă ce se urmăreşte ca rezultat.

5.5.2. Exemple de reprezentare a problemelor utilizînd calculul cu predicate de ordinul întîi

Exemplu 1: Se consideră problema traversării rîului de către un grup de excursionişti cu formularea: un grup de excursionişti ajung pe malul stîng al unui rîu ce nu poate fi traversat înot. La acelaşi mal se găsesc doi copii cu o barcă. În barcă poate să fie la un anumit moment un singur excursionist sau doi copii. Copiii sînt suficient de puternici ca să vîslească de mai multe ori în scopul traversării rîului. Starea finală în care se doreşte să se ajungă prevede ca grupul excursioniştilor să se găsească pe malul drept, iar cei doi copii împreună cu barca pe malul stîng.

Se propune următoarea grupare de simboluri:

- *suportul obiectual* format din mulţimea copiilor $C = \{c_1, c_2\}$, mulţimea excursioniştilor $E = \{e_1, e_2, ..., e_n\}$, elementul barcă $\{b\}$, cele două maluri $\{d, s\}$;
- *simboluri predicative* ce asertează următoarele proprietăţi:
  $C(x)$ cu semnificaţia că x este un copil, $x \in C$
  $E(x)$ cu semnificaţia că x este un excursionist, $x \in E$
  $D(x)$ cu semnificaţia că x este malul drept
  $S(x)$ cu semnificaţia că x este malul stîng
  $B(x)$ cu semnificaţia că barca este una din poziţiile $x \in \{s, d\}$
  $T(x, y, z)$ cu semnificaţia că x traversează cu barca din poziţia y în poziţia z în care, $x \in CUE$, $y, z \in \{s, d\}$;
- *faptele* sînt organizate în fapte relevînd condiţii ale problemei şi fapte relevînd proprietăţi ale obiectelor.

Se dau mai jos condiţiile şi expresiile formale ce definesc problema traversării:

- barca ajunsă pe malul drept trebuie readusă pe malul stîng numai de către un copil, un excursionist ajuns pe malul drept va rămîne acolo ştiind că numai aşa obiectivul problemei poate fi atins. Condiţia de acest tip se scrie:

$$[D(x) \wedge C(x)] \wedge B(d) \rightarrow T(x, d, s)$$

- dacă barca se găseşte pe malul stîng şi pe malul drept există un copil care să o readucă înapoi atunci un excursionist poate traversa rîul

$$[S(x) \wedge E(x)] \wedge [D(y) \wedge C(y)] \wedge B(s) \rightarrow T(x, s, d)$$

- dacă cei doi copii se găsesc pe malul stîng aceştia pot traversa rîul împreună utilizînd barca, fapt ce se exprimă prin formula:

$$[S(x) \wedge C(x)] \wedge [S(y) \wedge C(y)] \wedge B(s) \rightarrow T(x, s, d) \wedge T(y, s, d)$$

- traversarea rîului de către copii sau excursionişti atrage după sine schimbarea poziţiei bărcii

$$T(x, y, z) \rightarrow B(z)$$

- în urma unei traversări obiectele traversate vor fi plasate pe malul destinaţie

$$T(x, s, d) \rightarrow D(x)$$
$$T(x, d, s) \rightarrow S(x)$$

- dacă ambii copii sînt pe malul drept unul dintre ei va trebui să aducă barca pe malul stîng

$$[D(x) \wedge C(x)] \wedge [D(y) \wedge C(y)] \wedge B(d) \rightarrow T(x, d, s)$$

Faptele ce exprimă proprietăţi ale obiectelor sînt date de mulţimea copiilor, mulţimea excursioniştilor şi de poziţia iniţială a acestora, adică:

$C(c_1)$, $C(c_2)$, $E(e_1)$, $E(e_2)$,..., $E(e_n)$, $S(c_1)$, $S(c_2)$, $S(e_1)$,..., $S(e_n)$, $B(s)$, $B(d)$.

◆ *obiectivele problemei* sînt date de poziţia finală a copiilor, bărcii şi excursioniştilor:

$$D(e_1) \wedge D(e_2) \wedge ... \wedge D(e_n); S(c_1) \wedge S(c_2) \wedge B(s)$$

Forma generală a formulelor în calculul cu predicate de ordinul întîi nu este convenabilă pentru aplicarea formulelor cunoscute în logică. În acest sens o serie de transformări sînt aplicate acestor formule pentru a fi aduse la forma convenabilă mecanismului de interpretare.

Exemplul 2: Se consideră în continuare problema aşezării cuburilor, pentru care modul de reprezentare a cunoaşterii a fost tratat în capitolul 4. Se va vedea în paragrafele următoare că pentru multe probleme ce se reprezintă în spaţiul stărilor, concepte de căutare în grafuri se vor utiliza pentru descrierea

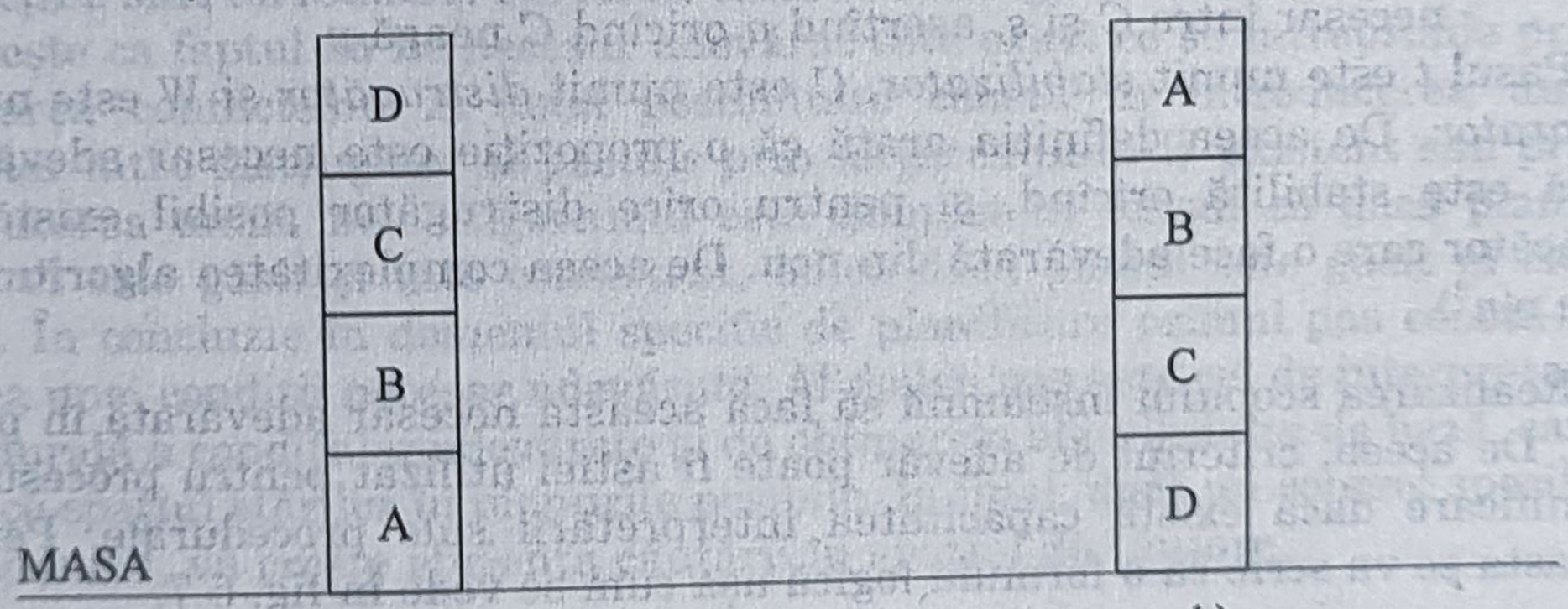
rezolvării. Green apropie spaţiul stărilor de formulele calculului cu predicate introducînd noţiunea de variabilă de stare. Ideea de bază porneşte de la descrierea ca stare a situaţiei iniţiale, a scopului ca o stare parţială şi a unor acţiuni ca operatori ce determină trecerea dintr-o stare în alta. Fiecare stare este reprezentată ca un set finit de formule în limbajul calculului cu predicate de ordinul întîi. Pentru simplitate se va admite că fiecare formulă este atomică. Prezenţa unei formule într-o stare arată că formula are valoarea adevărat. Fiecare acţiune este reprezentată printr-o condiţie de aplicabilitate sau precondiţie, adică, o acţiune nu este permisă a se executa în orice stare. Raţionamentul se specifică prin două liste, add_list ce reţine acele formule ce au valoarea adevărat şi delete_list pentru formulele ce nu mai sînt adevărate în noua stare. Lumea blocurilor este compusă dintr-o masă infinită în sensul că pot fi dispuse oricîte cuburi pe ea. Fiecare bloc are un nume ce este reprezentat în exemplu ca litere ale alfabetului. Starea din fig. 5.6a este descrisă prin următoarele formule:

$$\text{PE\_MASA}(A) \wedge \text{PE}(B, A) \wedge PE(C, B) \wedge PE(D, C) \wedge \text{CLEAR}(C)$$

Starea finală din fig. 5.6b va fi descrisă:

$$\text{PE\_MASA}(D) \wedge \text{PE}(C, D) \wedge \text{PE}(B, C) \wedge \text{PE}(A, B) \wedge \text{CLEAR}(A)$$

▼ ***Fig. 5.6 a) Starea iniţială a cuburilor***
b) Starea finală a cuburilor



Se vor considera în continuare trei acţiuni posibile, adică trei operatori:
UNSTACK - aşezarea cubului din vîrful stivei pe masă;
STACK - aşezarea unui cub aflat pe masă în vîrful stivei;
MOVE - mutarea unui cub dintr-o stivă în alta.
Acţiunile de acest tip se pot descrie prin

```
Unstack(x)
  Precondiţii:   PE(x, y), CLEAR(x)
  Add_list:      PE_MASA(x)
  Delete_list:     PE(x, y)
Stack(x, y)
```

```
   Precondiţii:   PE_MASA(x), CLEAR(x)
   Add_list:      PE(x, y)
   Delete_list:     PE_MASA(x), CLEAR(y)
Move(x, y)
   Precondiţii:   PE(x, z), CLEAR(x), CLEAR(y)
   Add_list:      PE(x, y), CLEAR(z)
   Delete_list:     PE(x, z), CLEAR(y)
```

Se poate uşor arăta că cea mai bună soluţie este dată de succesiunea operatorilor (Unstack(D), Move(C, D), Move(B, C), Stack(A)).

O altă idee este de a considera un plan neliniar cu scopuri nerealizate şi acţiuni parţial descrise ca reprezentări implicite a setului de planuri complete (complet ordonate, cu acţiuni descrise complet şi scopuri neatinse). Se va spune că propoziţia p este necesar adevărată (scris $\Box p$) în pasul s dacă şi numai dacă p este adevărată în s la fiecare completare a planului. Similar, o propoziţie p este posibil adevărată (scris $\Diamond p$) în pasul s dacă şi numai dacă p este adevărat în s la cel puţin o completare a planului. Aceleaşi definiţii sînt aplicabile în succesiunea paşilor spunînd că pasul s_1 este necesar înainte de s_2 (scris $\Box (s_1 < s_2)$), dacă este cazul la fiecare completare şi similar pentru posibilitate.

Cu restricţiile descrise, criteriul pentru necesar adevărat se reprezintă simplu. Astfel p este necesar adevărat în pasul s dacă şi numai dacă:

1. există un pas t după s în care p este necesat adevărat;
2. pentru toţi paşii C posibili, înainte de s şi negînd p există un pas W necesar între C şi s, asertînd p oricînd C neagă p.

Pasul t este numit *stabilizator*, C este numit *distrugător* şi W este numit *reparator*. De aceea definiţia arată că o propoziţie este necesar adevărată dacă este stabilită oricînd, şi pentru orice distrugător posibil există un refăcător care o face adevărată din nou. De aceea complexitatea algoritmului este $o(n^3)$.

Realizarea scopului înseamnă să facă aceasta necesar adevărată în pasul dat. De aceea, criteriul de adevăr poate fi astfel utilizat pentru procesul de planificare dacă există capacitatea interpretării sale procedurale. Pentru aceasta se va scrie ca o formulă logică aşa cum se vede în fig. 5.7.

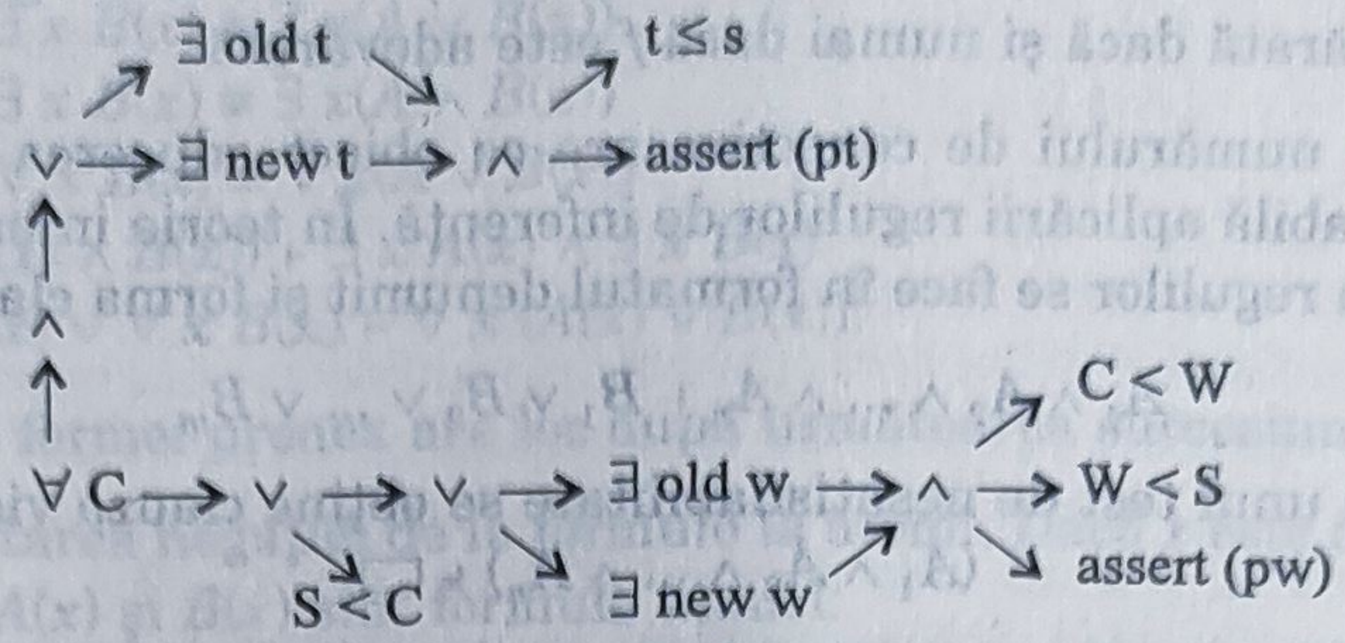
▼ ***Fig. 5.7. Formula logică***

$$\begin{array}{rl} \exists t \;\; t \le s \wedge \Box \text{ assert } (p,t) & \wedge \\ \forall c \; \Box \; s \le c & \vee \\ \forall q \; \Box \text{ negate } (c,p) & \vee \\ \exists w \; \Box \; c < w < s & \wedge \\ \text{assert } (p,w) & \end{array}$$

Conectivele au următoarea interpretare:

$\wedge$: ambele drumuri trebuie să fie adevărate;
$\vee$: alegere nedeterministă între cele două căi;
$\exists$: alegerea unui element, fie unul existent fie unul nou;
$\forall$: o buclă cu toate posibilităţile.

▼ *Fig. 5.8. Schiţa algoritmului*



Algoritmul rezultat este schiţat în fig. 5.8.

Ramura cea mai mare constă în găsirea unui pas existent asertînd *p* şi acesta să fie după *s*. Altă posibilitate este să se introducă un nou pas ce corespunde la rafinarea unei acţiuni de atingere a unui fapt dorit. Ca efect, precondiţiile sînt introduse ca noi obiective făcute necesar adevărate înainte de trecerea de la un pas la altul.

Cea mai scurtă ramură asigură că nu este o interacţie posibilă sau că interacţiile sînt corectabile. Forţarea tuturor distrugătorilor după pasul unde se doreşte ca faptul să fie necesar adevărat este exact ce se întreprinde prin rezolvarea conflictelor. A doua posibilitate constă în introducerea unui refăcător între distrugător şi pasul *s* prin forţarea unui *W* existent sau prin introducerea unuia nou. Algoritmul este complet în sensul că dacă planul există el este găsit şi este consistent, adică dacă planul este găsit el este corect. În concluzie în domeniul specific de planificare primul pas constă în găsirea unei condiţii necesar adevărate. Al doilea pas este cel de interpretare procedurală a condiţiilor adevărate şi de obţinere a algoritmului de bază, care produce spaţiul stărilor în planurile posibile. În final, dacă un control specific este adăugat, va creşte eficienţa căutării în spaţiul de căutare.

5.6. Skolemizarea formulelor

Activitatea de uniformizare a modului de scriere a formulelor se realizează în principal după următoarele etape:

- reducerea numărului de conective prin eliminarea implicaţiei şi echivalenţei;
- aducerea formulelor la forma normală prenex (cuantificatorii sînt aduşi înaintea corpului formulei pe care o prefixează);
- rescrierea formulei ca formulă universală prin eliminarea cuantificatorilor existenţiali utilizînd funcţiile Skolem;

- rescrierea formulelor utilizînd forma Skolem conjunctivă, mult mai convenabilă pentru demonstrarea automată.

Se spune că o formulă f are o formă Skolem f' ce nu este logic echivalentă, dar este adevărată dacă şi numai dacă f este adevărată.

Reducerea numărului de conective are ca obiect aducerea formulelor la forma convenabilă aplicării regulilor de inferenţă. În teoria inferenţelor logice reprezentarea regulilor se face în formatul denumit şi forma clauzală de tipul

$$A_1 \wedge A_2 \wedge \ldots \wedge A_n \vdash B_1 \vee B_2 \vee \ldots \vee B_m$$

În situaţia unui test de nesatisfiabilitate se obţine clauza vidă de forma

$$(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \vdash \square$$

Considerînd că $A_1, A_2, \ldots, A_m$ sînt formule ce se presupun a fi independente, pentru fiecare există o clauză A_i $1 <= i <= n$. Se dau mai jos cîteva reguli de calcul în formule ale limbajului de calcul cu predicate de ordinul întîi

$(A \to B) \vdash (\neg A \vee B)$
$\neg(A \to B) \vdash (A \wedge \neg B)$
$(A = B) \vdash (A \vee \neg B) \wedge (\neg A \vee B)$.

Fie x o variabilă şi A, B, $A(x)$ şi $B(x)$ formule, în care A şi B nu conţin apariţii libere ale variabilei x, atunci:

$\vdash \forall x\,(A \to B(x)) \equiv A \to \forall x B(x)$
$\vdash \forall x\,(A(x) \to B) \equiv \exists x A(x) \to B$
$\vdash \exists x\,(A \to B(x)) \equiv A \to \exists x B(x)$
$\vdash \exists x\,(A(x) \to B) \equiv \forall x A\,(x) \to B$

Pornind de la o formulă oarecare A se poate găsi o formulă A' numită şi formă prenex a lui A cu proprietăţile:

- A' este echivalentă cu A adică $\vdash A \equiv A'$;
- în formula A', cuantificatorii sînt plasaţi în partea cea mai din stînga a formulei prefixînd corpul formulei în care se găsesc simbolurile conectivelor logice ce figurează în scopul oricărui cuantificator.

Reguli de transformare. Dacă x şi y sînt variabile distincte, A, B, $A(x)$, $B(x)$, $A(x, y)$ formule, iar A şi B nu conţin apariţii libere ale lui x şi dacă x este liber pentru y în $A(x, y)$ atunci:

$\vdash \forall x A \equiv A$
$\vdash \exists x A \equiv A$
$\vdash \forall x \forall y A(x, y) \equiv \forall y \forall x A(x, y)$
$\vdash \exists x \exists y A(x, y) \equiv \exists y \exists x A(x, y)$
$\forall x \forall y A(x, y) \vdash \forall x A(x, x)$
$\exists x A(x, x) \vdash \exists x \exists y A(x, y)$
$\forall x A(x) \vdash \exists x A(x)$
$\exists x \forall y A(x, y) \vdash \forall y \exists x A(x, y)$
$\vdash \exists x A(x) \equiv \neg\exists x \neg A(x)$
$\vdash \forall x A(x) \equiv \neg\forall x \neg A(x)$
$\vdash \neg\forall x A(x) \equiv \exists x \neg A(x)$

$\vdash \rceil \exists\, x\, A(x) \equiv \forall\, x\, \rceil A(x)$
$\vdash \forall\, x\, A(x) \wedge \forall\, x\, B(x) \equiv \forall\, x(A(x) \wedge B(x))$
$\vdash \exists\, x\, A(x) \vee \exists\, x\, B(x) \equiv \exists\, x(A(x) \vee B(x))$
$\vdash A \wedge \forall\, x\, B(x) \equiv \forall\, x(A \wedge B(x))$
$\vdash A \vee \exists\, x\, B(x) \equiv \exists\, x(A \vee B(x))$
$\vdash A \wedge \exists\, x\, B(x) \equiv \exists\, x(A \wedge B(x))$
$\vdash A \vee \forall\, x\, B(x) \equiv \forall\, x(A \vee B(x))$
$\exists\, x\, (A(x) \wedge B(x)) \vdash \exists\, x\, A(x) \wedge \exists\, x\, B(x)$
$\forall\, x\, A(x) \vee \forall\, x\, B(x) \vdash \forall\, x\, (A(x) \vee B(x))$.

Obţinerea formei prenex are loc după următoarea succesiune de operaţii:

- tranzitarea negaţiei de la formule la atomi. Dacă x este o variabilă, iar A, B, $A(x)$ şi $B(x)$ sînt formule atunci:
  $\rceil \forall\, x\, A(x) \equiv \exists\, x\, \rceil A(x)$
  $\rceil \exists\, x\, A(x) \equiv \forall\, x\, \rceil A(x)$
  $\rceil (A \wedge B) \equiv \rceil A \vee \rceil B$
  $\rceil (A \vee B) \equiv \rceil A \wedge \rceil B$
  $\rceil \rceil A \equiv A$;
- transferul cuantificatorilor de la formule la atomi. Dacă x este o variabilă $A(x)$ şi B sînt formule cu x liberă în A dar legată în B atunci:
  $\forall\, x\, (A(x) \vee B) \vdash \forall\, x\, A(x) \vee B$
  $\forall\, x\, (B \vee A(x)) \vdash B \vee \forall\, x\, A(x)$
  $\forall\, x\, (A(x) \wedge B) \vdash \forall\, x\, A(x) \wedge B$
  $\forall\, x\, (B \wedge A(x)) \vdash B \wedge \forall\, x\, A(x)$
  $\exists\, x\, (A(x) \vee B) \vdash \exists\, x\, A(x) \vee B$
  $\exists\, x\, (B \vee A(x)) \vdash B \vee \exists\, x\, A(x)$
  $\exists\, x\, (B \wedge A(x)) \vdash B \wedge \exists\, x\, A(x)$
  $\exists\, x\, (A(x) \wedge B) \vdash \exists\, x\, A(x) \wedge B$.

Dacă x este o variabilă, iar $A(x)$ şi $B(x)$ sînt formule cu x liberă atît în $A(x)$ cît şi în $B(x)$ atunci:
$\forall\, x\, (A(x) \wedge B(x)) \vdash \forall\, x\, A(x) \wedge \forall\, x\, B(x)$
$\exists\, x\, (A(x) \vee B(x)) \vdash \exists\, x\, A(x) \vee \exists\, x\, B(x)$.

Dacă x şi y sînt variabile $A(x, y)$ şi $B(x, y)$ sînt formule, x este legată în cel puţin una din formulele $A(x, y)$ şi $B(x, y)$, iar y este liberă atît în $A(x, y)$ cît şi în $B(x, y)$, atunci:
$\forall\, x\, \forall\, y\, (A(x, y) \vee B(x, y)) \vdash \forall\, y\, \forall\, x\, (A(x, y) \vee B(x, y))$
$\exists\, x\, \exists\, y\, (A(x, y) \wedge B(x, y)) \vdash \exists\, y\, \exists\, x\, (A(x, y) \wedge B(x, y))$;

- schimbarea numelor variabilelor. Cînd doi cuantificatori prefixează acelaşi nume de variabilă, numele variabilei este înlocuit cu un simbol diferit de celelalte simboluri pentru variabilele utilizate. Procedura se aplică în mod repetat pînă se obţine individualizarea simbolurilor pentru variabile în formula considerată. Procedeul se mai numeşte şi standardizarea variabilelor;

- eliminarea cuantificatorului existenţial. Acest proces se bazează pe înlocuirea variabilelor cuantificate de acesta prin funcţii Skolem.

Fie f o formulă în limbajul calculului cu predicate de ordinul întîi avînd variabilele libere $x_1, x_2, \ldots, x_n, y$. *Pentru o variabilă oarecare, (de exemplu y) se poate asocia un simbol funcţional* $f_f(x_1, x_2, \ldots, x_n)$ *denumit şi funcţie Skolem a lui f ce satisface axioma Skolem a lui f.*

$\vdash x_1, x_2, \ldots, x_n (\exists y f(x_1, x_2, \ldots, x_n, y) \rightarrow f(x_1, x_2, \ldots, x_n, f_f(x_1, x_2, \ldots, x_n))$.

Astfel se elimină din formula dată cuantificatorul existenţial prin înlocuirea variabilei cuantificată cu un simbol funcţional inexistent avînd ca argumente acele variabile ce sînt cuantificate universal. Se remarcă faptul că simbolurile de variabile cuantificate existenţial, ce nu figerează în scopul nici unui cuantificator universal, reprezintă funcţii 0-are. Astfel formula $\exists v \forall x \forall y \exists z P(x, y, z, v)$ este echivalentă cu $\forall x \forall y P(x, x, f(x, y), a)$, în care, variabila z cuantificată existenţial a fost înlocuită cu funcţia Skolem $f(x, y)$ cu argumentele cuantificate universal în al căror scop se află cuantificarea existenţială $\exists z$, variabiala v cuantificată existenţial a fost înlocuită cu o funcţie Skolem $f(\ ) = a$ (constantă), întrucît cuantificarea existenţială $\exists v$ nu se află în scopul nici unei cuantificări universale, aşa că funcţia Skolem este de aritate zero;

- tranzitarea în prefix a cuantificatorilor universali. Întrucît simbolurile de variabile sînt individualizate pot fi grupaţi în prefix toţi cuantificatorii universali obţinînd astfel *forma normală prenex* a formulei. Un cuantificator ce are ca scop o subformulă are acelaşi scop şi dacă prefixează întreaga formulă.

În concluzie, o formulă în forma normală prenex este alcătuită din două părţi, şi anume:

- partea din stînga ce a fost denumită şi prefix, şi cuprinde toţi cuantificatorii universali ai formulei;
- partea plasată după prefix numită şi matricea formulei ce cuprinde literale libere de cuantificatori şi alcătuieşte corpul propriu-zis al formulei.

Forma normală prenex permite transcrierea subformulelor componente în vederea obţinerii formei normale conjunctive, numită şi forma Skolem conjunctivă, formată din conjuncţii de disjuncţii. Dacă A, B, C sînt formule, atunci:

$\vdash A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$

$\vdash (A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$

Astfel, forma Skolem conjunctivă trebuie eventual simplificată pentru a se ajunge la cea mai condensată formă de exprimare a formulei. Procedura ce duce la forma condensată implică următorii paşi:

- se elimină tautologiile;
- dacă se întîlnesc subformule de tipul $\alpha \vee A \vee \alpha$, cu α un literal şi A o formulă este echivalentă cu $\alpha \vee A$;
- dacă A, B şi C sînt clauze cu $\forall B$ şi $\forall C$ închiderea universală a clauzelor B şi C adică $\vdash \forall B \rightarrow \forall C$ se pot aplica regulile de

transcriere $B \wedge A \wedge C \vdash B \wedge A$ şi $C \wedge A \wedge B \vdash A \wedge B$, şi despre clauza C se spune că este subsumată clauzei B.

Transcrierea în notaţie de mulţime clauzală se face pentru o utilizare mai uşoară a formulelor Skolem aplicînd criterii de retranscriere. În acest scop cuantificatorii universali ce prefixează formula sînt şterşi şi sînt consideraţi implicit pentru toate simbolurile de variabilă, variabilele sînt legate şi prin standarizarea simbolurilor ordinea în care sînt înscrişi nu mai este relevantă la interpretare. Simbolurile $\vee$ sînt omise, simbolurile $\wedge$ sînt înlocuite prin virgule, parantezele cu excepţia celor pentru variabilele funcţiilor sînt omise. Formula $\forall x \forall y \forall z ((P(x) \vee Q(y)) \wedge R(z))$ se va scrie $Px\ Qy, Rz$ notaţie ce se numeşte şi mulţime cauzală.

5.7. Reprezentarea în spaţiul stărilor

Reprezentarea problemei în spaţiul stărilor presupune luarea în consideraţie a stărilor şi transformărilor acestora datorate unui număr finit de operatori. Matematic se poate considera scrierea ca reprezentată prin tripletul $P = (S, G, R)$ în care,

- S reprezintă mulţimea stărilor;
- $G \subset S$ mulţimea obiectivelor problemei;
- $R \subset SxS$ mulţimea de transformări de stare ce indică un drum în *"graful problemei"*.

Se spune că secvenţa de stări $(s_0, s_1, ..., s_n)$ formează un drum în graf dacă $s_i s_{i+1} \in R\ (\forall)\ i = \overline{0,n}$, iar n-uplul $(s_0, s_1, ..., s_n)$ reprezintă o soluţie a problemei pentru s_0, dacă $s_n \in G$.

Cunoaşterea despre problemă este definită prin:

- mulţimea obiectelor abstracte ale problemei;
- mulţimea de operatori;
- starea iniţială;
- starea finală formată din stări ce reprezintă obiectivul problemei.

Se defineşte obiectivul problemei, ca o structură simbolică ce reprezintă o stare de lucruri care odată atinsă confirmă faptul că drumul ce a fost parcurs de la starea iniţială prin graful problemei este o soluţie. Un alt mod de a privi obiectivul este ca o procedură pentru testarea comportării ce oferă criterii de apreciere în căutarea soluţiilor prin intermediul spaţiului stărilor. În procesul de rezolvare a problemei, starea problemei reprezintă o informaţie dinamică cu un conţinut care evoluează ca urmare a mecanismelor de rezolvare. Pot fi stabilite cîteva obiective în procesul rezolvării, şi anume:

- parcurgerea înainte în spaţiul stărilor cu scopul atingerii obiectivului;

- parcurgerea înapoi avînd ca scop revenirea la punctele anterioare pentru comutarea procesului de rezolvare pe o altă alternativă de drum.

Informaţia de stare conţine acele elemente ale cunoaşterii prin care:

- se asigură tranziţia de la o stare la alta, înainte şi înapoi;
- se asigură informaţia de acces la memorie pentru oricare din stările curente;
- se asigură informaţia de revenire pentru refacerea stării sistemului în punctele de ramificaţie importante ale grafului şi selectarea unei alternative în caz de eşec;
- se asigură informaţia de acces la alte trasee ale grafului în situaţia prelucrărilor paralele;
- păstrarea informaţiilor de justificare a paşilor sistemului de rezolvare.

Prin spaţiul stărilor se înţelege o mulţime finită de obiecte implicate în rezolvarea problemei pornind de la starea iniţială prin aplicarea unei mulţimi finite de operatori. Aplicarea unui operator asupra unei stări are ca efect producerea unei stări noi sau a unui număr finit de stări alternative. Rezolvarea presupune căutarea în spaţiul stărilor a unei stări ce corespunde obiectivului problemei. Parcurgerea grafului de la starea iniţială s_0 la starea finală s_n se face prin alegerea unui drum dintr-un număr finit de alternative. În acest proces de parcurgere este cunoscut criteriul de apreciere a obiectivului fără a se cunoaşte ordinea de aplicare a operatorilor sau lungimea drumului de parcurs.

Se consideră ca exemplu o zonă închisă de 9 spaţii în care sînt aşezate un număr de 8 cutii şi un spaţiu gol. Se cere să se găsească modul de mişcare a cutiilor în zona delimitată cu un singur spaţiu liber pentru ca o anumită cutie să ajungă dintr-o poziţie iniţială dată într-o poziţie finală dată, celelalte cutii găsindu-se în poziţii nestabilite. Presupunînd descrierea stării sub forma matricială în care a fost notată cu *e* poziţia golului starea iniţială a problemei este dată în fig. 5.9a iar starea finală în fig. 5.9b.

▼ ***Fig. 5.9 a) Starea iniţială***
b) Starea finală

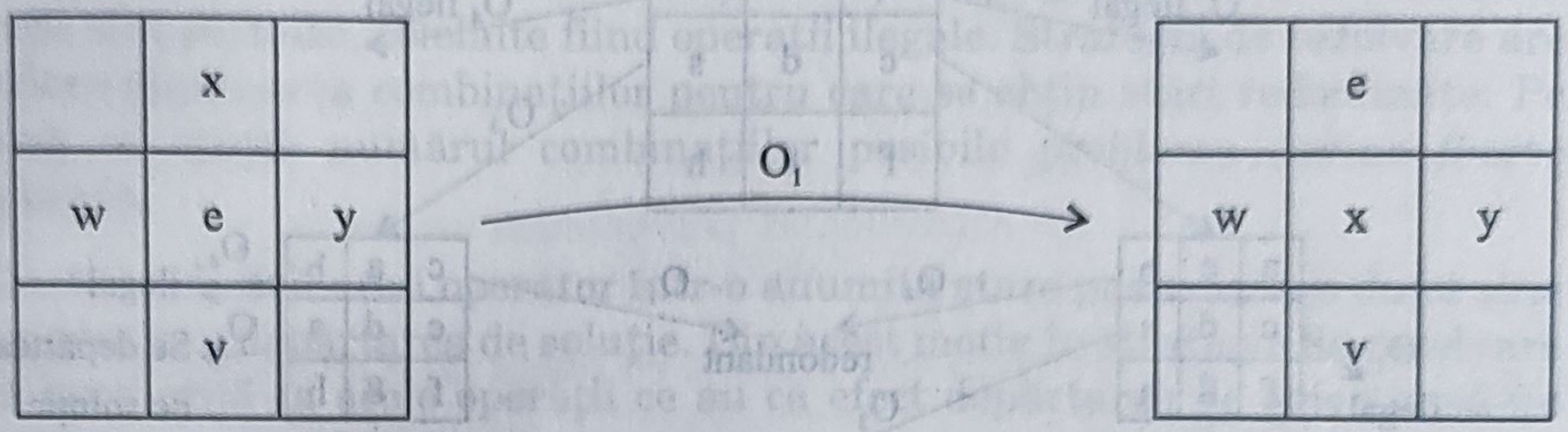
a)

i \ j	1	2	3
1	e	a	b
2	c	d	s
3	f	g	h

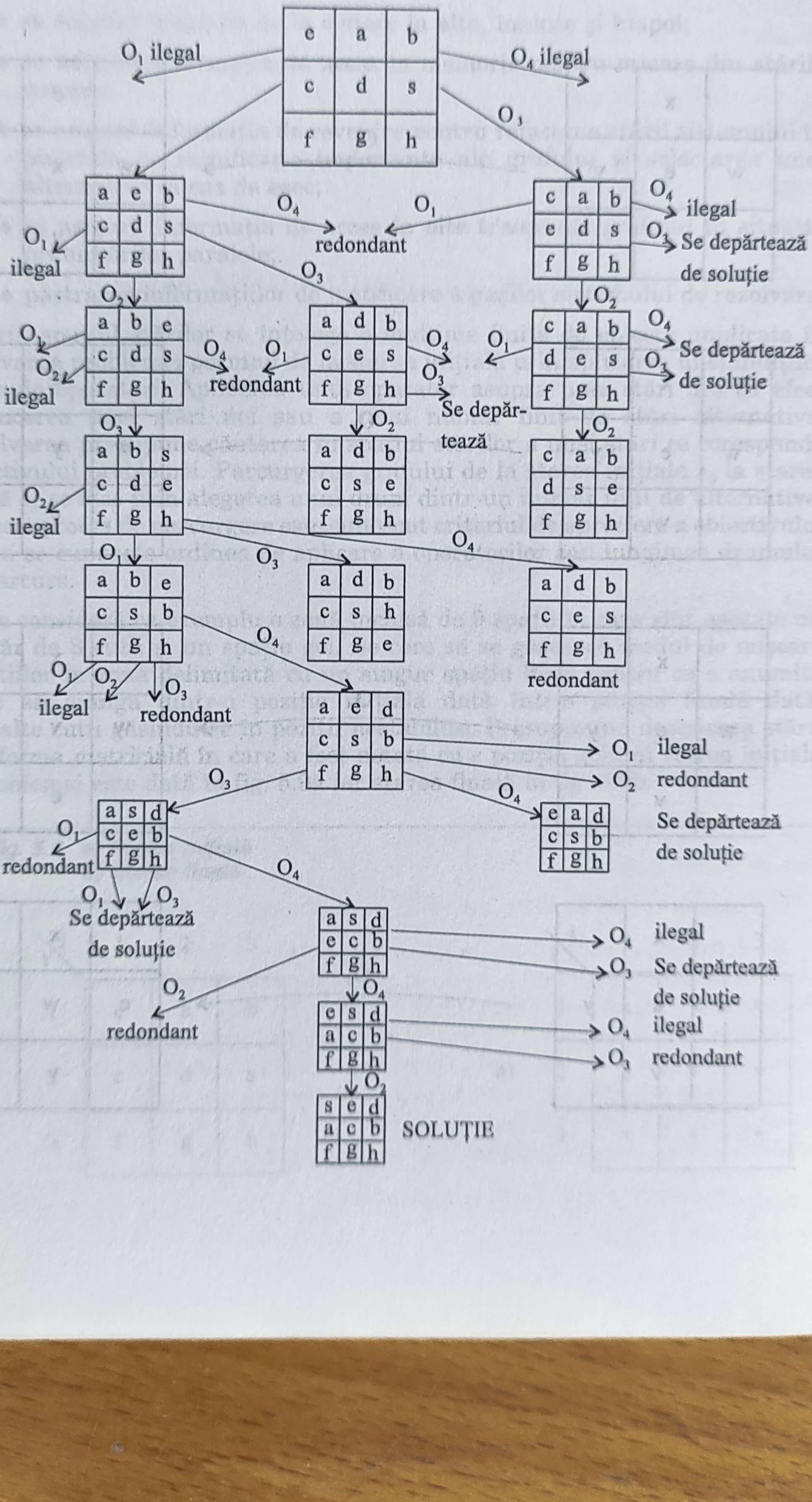
b)

i \ j	1	2	3
1	s	*	*
2	*	*	*
3	*	*	*

▼ ***Fig. 5.10. Operatori***



▼ **Fig. 5.11. Paşi de rezolvare a problemei**



Obiectivul problemei este reprezentat de starea finală a cutiei *s*, poziţiile celorlalte cutii şi a golului sînt marcate prin *.

În strategia de rezolvare se consideră patru operatori notaţi O_1, O_2, O_3, O_4 pentru deplasarea golului, operatori ce sînt explicitaţi în spaţiul stărilor ca în fig. 5.10. Într-o anumită stare funcţie de poziţia golului numai anumite operaţii sînt permise, celelalte fiind operaţii ilegale. Strategia de rezolvare are în vedere eliminarea combinaţiilor pentru care se obţin stări redondante. Pe măsură ce creşte numărul combinaţiilor posibile problema devine foarte complicată.

Efectul aplicării unui operator într-o anumită stare poate atrage după sine apropierea sau depărtarea de soluţie. Din acest motiv în strategia de rezolvare se va avea grijă ca acele operaţii ce au ca efect depărtarea de obiectiv să fie rejectate.

În fig. 5.11 se dau cîteva nivele ale rezolvării problemei evidenţiind la fiecare pas acei operatori ce au ca efect depărtarea de soluţie sau revenirea la o stare redondantă. De asemenea se observă că sînt mai multe soluţii valide iar rezultatul se găseşte în număr diferit de paşi. Din analiza soluţiilor obţinute se evidenţiază următoarele secvenţe de operatori:

$O_2O_3O_2O_1O_4O_3O_4O_1O_2$ 9 paşi
$O_3O_2O_2O_1O_4O_3O_4O_1O_2$ 9 paşi
$O_2O_2O_3O_1O_4O_3O_4O_1O_2$ 11 paşi

Toţi cei trei arbori sînt corecţi din punct de vedere logic însă au adîncimi diferite. Raţionamentul în spaţiul stărilor apare ca o modalitate de a capta raţionamentul productiv, apărînd noi stări printre care se caută obiectivul problemei. Reducerea numărului de combinaţii ale reprezentării se face prin introducerea regulilor restrictive de limitare a combinaţiilor.

Modul de reprezentare în spaţiul stărilor este posibil a se utiliza şi în cazul raţionamentelor reductive, avînd ca efect obţinerea unei subprobleme sau a unei mulţimi finite de subprobleme. O altă variantă de reprezentare este cea prin care operatorii sînt văzuţi ca reguli de infereţă, se aplică asupra clauzelor avînd ca efect producerea altor clauze. Structura arborescentă ce se obţine ridică o serie de probleme cum sînt:

- definirea regulilor restrictive pentru procesul generator de stări avînd ca scop eliminarea stărilor redondante, a celor care se depărtează de soluţie, a combinaţiilor absurde ce nu conduc la soluţie;
- definirea unor strategii de căutare a soluţiei în spaţiul generat;
- definirea unei strategii de revenire în caz de eşec;
- alegerea unei reprezentări compacte ca spaţiu de memorie ocupat.

Se prezintă mai jos problema traversării rîului pentru a cărei rezolvare se va utiliza spaţiul stărilor. Notînd cu S şi D stări parţiale ale problemei ce reprezintă situaţia pe cele două maluri, stîng şi respectiv drept, se obţine starea ca o reuniune a celor două stări parţiale $S \cup D$.

starea iniţială: $S_0 = \{c_1, c_2, e_1, e_2, \ldots, e_n, b\}$
$D_0 = \varnothing$
starea finală: $S_f = \{c_1, c_2, b\}$
$D_f = \{e_1, e_2, \ldots, e_n\}$
operatori: Ted - un copil traversează cu barca pe malul drept $\{c_j, b\} \subset s_i$, $j = 1, 2$
T2cd - doi copii traversează cu barca pe malul drept $\{c_1, c_2, b\} \subset s_i$
Ted - un excursionist traversează cu barca pe malul drept $\{e_j, b\} \subset s_i$, $\overline{j=1,n}$
Tcs - un copil traversează cu barca pe malul stîng $\{c_j, b\} \subset d_i$, $j = 1, 2$
T2cs - doi copii traversează cu barca pe malul stîng $\{c_1, c_2, b\} \subset d_i$
Tes - un excursionist traversează cu barca pe malul stîng $\{e_j, b\} \subset d_i$, $\overline{j=1,n}$.

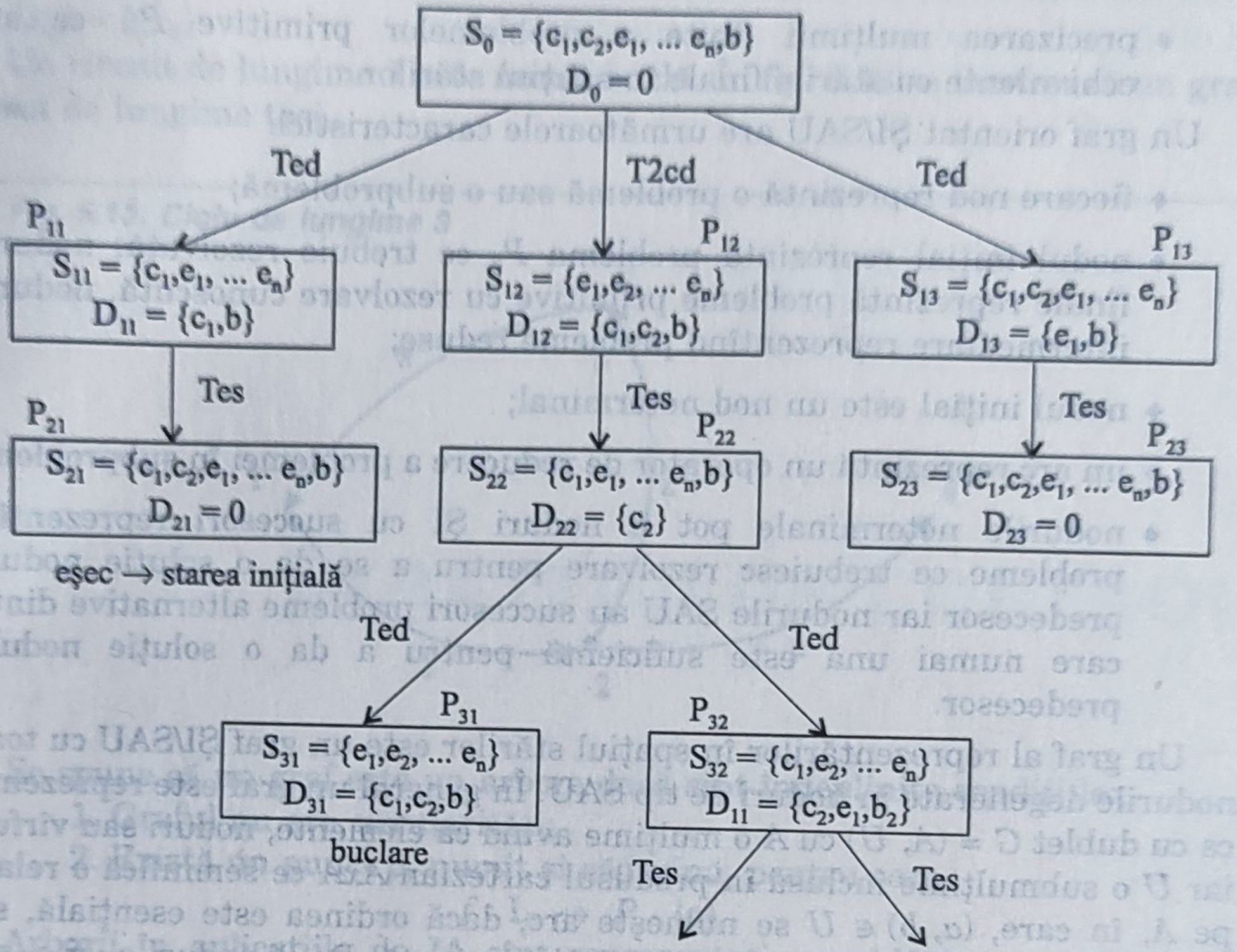
Prin aplicarea operatorilor definiţi mai sus se poate genera spaţiul stărilor, spaţiu în care prima stare este starea iniţială. Similar cu problema aşezării cutiilor în spaţiul închis anumiţi operatori sînt invalizi în sensul că nu pot fi aplicaţi stării curente, alţii conduc la stări redondante sau se depărtează de soluţie. În fig. 5.12 se arată modul de formare a arborelui asociat problemei de traversare a rîului pentru primele trei nivele. Graful de tranziţie a stărilor cu punctul de plecare stare iniţială are ramuri ce conduc la obiectivul final, operatorii ce conduc la stări redondante fiind eliminaţi. Soluţia problemei poate fi privită fie ca o succesiune de stări fie ca o secvenţă a operatorilor ce conduc la starea obiectiv. Furnizarea soluţiei ca o secvenţă de operatori ce conduce la obiectiv va da

$$O = \{\text{T2cd, Tcs, Ted, Tcs, T2cd, Tcs, Ted, Tcs, T2cd,...}\}$$

în care secvenţa $O' = \{\text{T2cd, Tcs, Ted, Tcs}\}$ se repetă de un număr de ori egal cu numărul excursioniştilor. Soluţia furnizată în acest mod este soluţia oricărei probleme de traversare cu doi copii, o barcă şi un număr oarecare de excursionişti.

Din analiza reprezentării în spaţiul stărilor se observă o asemănare între aceasta şi reprezentarea sistemelor de producţii. Astfel, un operator se aplică numai dacă anumite criterii prealabile sînt îndeplinite de starea asupra căreia se aplică operatorul (echivalentă cu precondiţia), rezultatul aplicării operatorului fiind obţinerea unei stări noi din cea veche (echivalentul acţiunii sau producţiei). Strategia de control este cea care determină selectarea operatorilor (regulilor de producţie) şi a modului de aplicare succesivă a acestora pentru generarea spaţiului stărilor.

▼ ***Fig. 5.12. Arbore asociat problemei traversării râului***



Aplicarea unui operator asupra unei stări fără a putea reveni la starea anterioară pentru a apela un alt operator formează *strategia de control irevocabilă*. Strategia de control se numeşte *tentativă* dacă la un anumit punct se poate lua decizia de revenire la o stare precedentă în vederea aplicării de noi operatori. Strategia de control cu *revenire* (backtraking) este acea strategie prin care punctul de reluare este stabilit din momentul selectării operatorului, aşa că în caz de eşec se poate relua procesul de la un astfel de punct, nu neapărat cel mai apropiat. Strategia de control prin *căutare de grafuri* este cea prin care se păstrează informaţii privind consecinţele aplicării operatorilor la fiecare pas al generării.

5.8. Utilizarea grafurilor ŞI/SAU

Aplicarea strategiilor de rezolvare reductive asupra unei probleme P_0 porneşte de la următoarele precizări:

- descrierea problemei iniţiale P_0 exprimată în spaţiul stărilor prin starea iniţială P_0;

- precizarea unei mulţimi finite de operatori care aplicaţi asupra problemei produc o mulţime finită de subprobleme;
- precizarea mulţimii finite a problemelor primitive P_p, ce sînt echivalente cu stările finale în spaţiul stărilor.

Un graf orientat ŞI/SAU are următoarele caracteristici:

- fiecare nod reprezintă o problemă sau o subproblemă;
- nodul iniţial reprezintă problema P_0 ce trebuie rezolvată, nodurile finale reprezintă probleme primitive cu rezolvare cunoscută, nodurile intermediare reprezentînd probleme reduse;
- nodul iniţial este un nod neterminal;
- un arc reprezintă un operator de reducere a problemei în subprobleme;
- nodurile neterminale pot fi noduri ŞI cu succesori reprezentînd probleme ce trebuiesc rezolvare pentru a se da o soluţie nodului predecesor iar nodurile SAU au succesori probleme alternative dintre care numai una este suficientă pentru a da o soluţie nodului predecesor.

Un graf al reprezentărilor în spaţiul stărilor este un graf ŞI/SAU cu toate nodurile degenerate în noduri de tip SAU. În general un graf este reprezentat ca cu dublet $G = (A, U)$ cu A o mulţime avînd ca elemente, noduri sau vîrfuri, iar U o submulţime inclusă în produsul cartezian AxA ce semnifică o relaţie pe A, în care, $(a, b) \in U$ se numeşte arc, dacă ordinea este esenţială, sau muchie dacă ordinea nodurilor nu este importantă. Două noduri a_i şi a_j sînt adiacente dacă există un arc sau o muchie ce leagă a_i de a_j. O secvenţă de noduri şi arce adiacente ce sînt parcurse în acelaşi sens formează un drum. Se poate deci defini drumul prin succesiunea nodurilor $d = (a_1, a_2,..., a_n)$ cu proprietatea că $(a_1, a_2),..., (a_{n-1}, a_n)$ sînt adiacente. Nodul a_1 este capătul iniţial al drumului iar nodul a_n este capătul final al drumului. Dacă $a_1 = a_n$ atunci drumul defineşte un circuit. Numărul de arce al unui drum defineşte lungimea drumului. Se numeşte drum elementar acel drum care trece o singură dată printr-un nod, iar drum simplu un drum ce nu conţine de două ori acelaşi arc. Pentru un graf neorientat se numeşte lanţ, o succesiune $(x_1, x_2,..., x_n)$, ciclu în lanţ dacă $x_1 = x_n$, lanţ elementar dacă nodurile sînt distincte, lanţ simplu dacă toate muchiile sînt distincte.

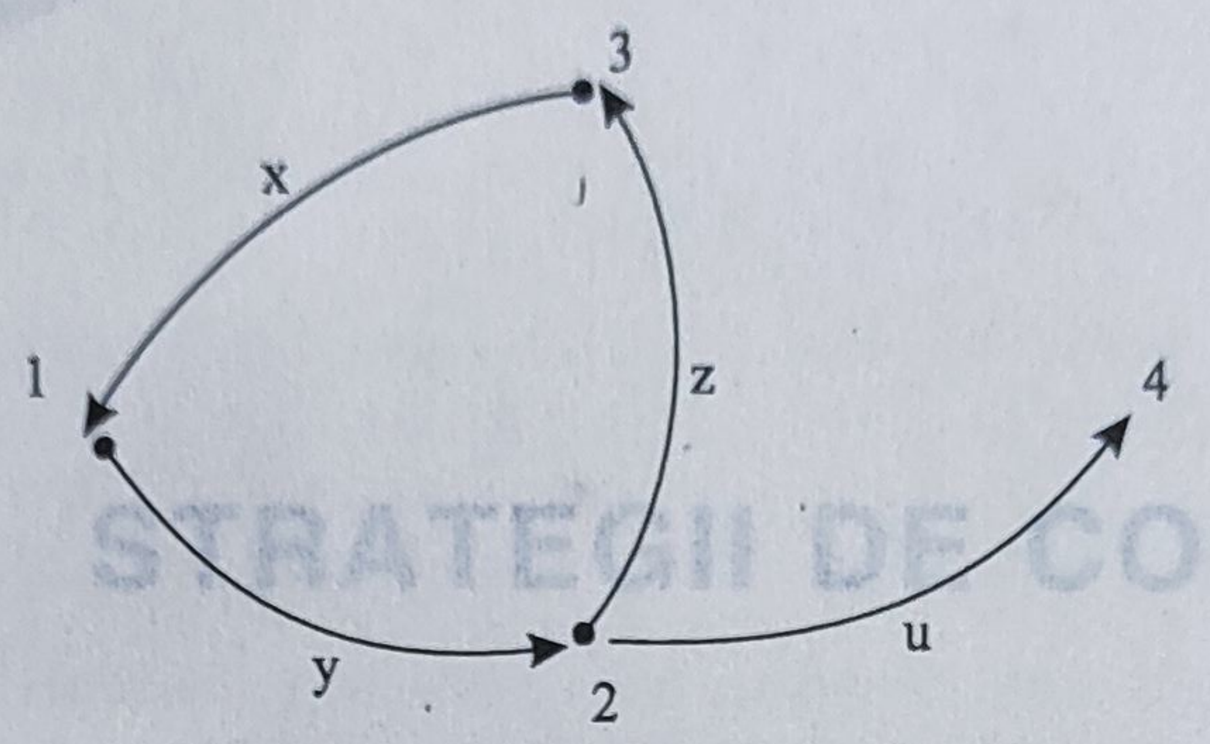
Se spune că un graf G este conex dacă oricare dintre două noduri sînt extremităţi ale unui lanţ în G. Un graf conex fără cicluri este un arbore. Într-un graf se numeşte arborescenta de rădăcină a, un graf orientat în care orice nod $a_i \# a$ este o extremitate pentru un drum unic în care cealaltă extremitate este a.

Fundamentală pentru discuţiile ulterioare este noţiunea de circuit într-un graf. Intuitiv, un circuit este o buclă închisă formată din arce care traversează în secvenţă punctele circuitului pornind de la un punct de pornire. Riguros, dacă (P, L, f, b) este un graf direct cu funcţia de înaintare f şi de revenire b, arcele distincte $x_1, x_2,..., x_n$ formează un circuit de lungime n dacă:

$f(x_1) = b(x_2)$
$f(x_2) = b(x_3)$
⋮
$f(x_n) = b(x_1)$

Un circuit de lungime 1 este numit buclă. În fig. 5.13 se ilustrează un graf direct de lungime trei.

▼ *Fig. 5.13. Ciclu de lungime 3*



Se spune că un graf este un arbore dacă sînt îndeplinite condiţiile:

1. Graful nu are circularităţi;
2. Există un punct p, numit şi rădăcină, pentru care

$$f : L \rightarrow P - \{p\}$$

Arborii în aplicaţiile de IA sînt reprezentaţi cu rădăcina sus şi arcele orientate în jos. Din acest motiv săgeţile se pot omite întrucît totdeauna sînt orientate de sus în jos. Frunzele arborelui sînt acele noduri q din P de la care nu pornesc arce spre alte noduri. Caracterizarea cea mai uzuală a adîncimii unui nod într-un arbore este nivelul nodului. Subsetul P_1 al punctelor $f(x_i)$ din P pentru care

$b(x_1) = b(x_2) = ... = b(x_n) = p$

constituie nuvelul 1. Atunci:

$P_1 = \{f(x_1), f(x_2),..., f(x_n)\}$.

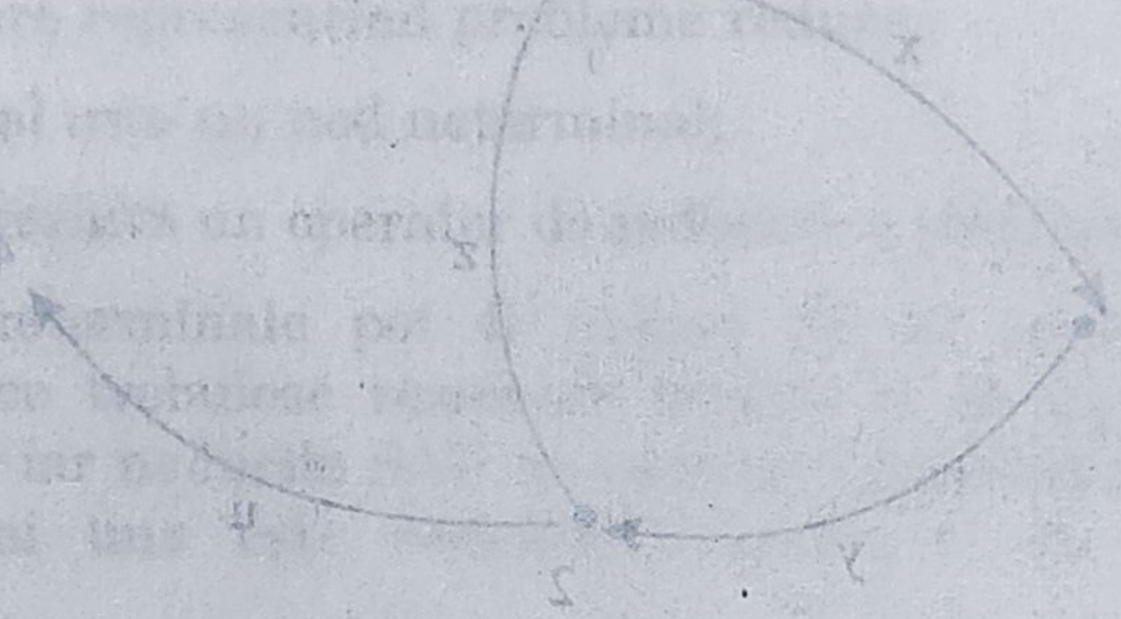
Setul P_k al punctelor nivelului k este definit recursiv funcţie de nivelul k - 1. Astfel, dacă $b(x_i) \in P_{k-1}$ pentru $i = 1, 2,..., j$ atunci:

$P_k = \{f(x_i): i = 1, 2,..., j\}$.

Este important de reţinut faptul că maşina priveşte graful altfel decît omul, în sensul că pentru maşină singurul acces la graf este asociat cvadruplului (P, L, f, b).

Ca exemplu, pentru calculul integralei definite se poate aplica tehnica descompunerii în serie, integrarea prin părţi, schimbarea de variabilă. Fiecare din aceste metode conduce la soluţie validă, deci pornind de la starea iniţială (problema P_0) un arbore SAU indică necesitatea obţinerii unei soluţii fără a fi necesară revenirea şi reluarea unei alte ramuri a arborelui. Rezolvarea după generarea arborelui ŞI/SAU de reducere a problemei presupune căutarea în graf a condiţiilor necesare ca nodul iniţial reprezentînd problema sa fie

solvabil. Se spune că un nod este solvabil dacă nodul este un nod terminal, deci o problemă primitivă, dacă nodul este un nod neterminal de tip ŞI ce are toate nodurile succesoare solvabile, dacă nodul este nod neterminal de tip SAU şi cel puţin un nod succesor este solvabil.



STRATEGII DE CONTROL

Sa arătat în capitolul precedent că o problemă reprezentată în spaţiul stărilor conduce la o rezolvare ce se reduce la căutare în spaţiul stărilor. Alegerea tehnicii de rezolvare este dependentă de tipul problemei, mai multe tehnici fiind posibil a fi aplicate. Cea mai simplă tehnică este cea de selecţie a operatorilor "trial-and-error process", tehnica ce nu este totdeauna o alternativă viabilă întrucît nu oferă o căutare sistematică în spaţiul stărilor. O altă soluţie este cea a atingerii costului minim, iar în situaţia în care costul atingerii obiectivului poate fi predictat, timpul de căutare este corespunzător scurtat prin utilizarea acestei informaţii. În acest capitol se vor analiza o parte dintre posibilii algoritmi ce căutare.

6.1. Căutarea exhaustivă (Trial-and-error searching)

Metoda cea mai simplă este cea de aplicare a tuturor operatorilor selectaţi pînă ce scopul este atins. Algoritmul asociat este prezent mai jos.

```
stabileşte starea iniţială
while (stare # starea obiectiv)
{
  selectează un operator aplicabil în această stare pe care îl
  aplică;
  stare =   operator (stare); /* aplicarea operatorului la stare
            produce o altă stare ce devine starea actuală */
}
```

În pasul de selecţie a operatorului, acesta este ales aleator dintre operatorii aplicabili. Procedura devine stochastică, însă nu se garantează atingerea obiectivului. Metoda simplă de acest tip este capabilă să soluţioneze probleme simple. Pentru a justifica modul în care se ajunge la soluţie schimbările succesive de stare sînt înregistrate în memorie.

6.2. Căutarea prin examinarea semantică a nodurilor

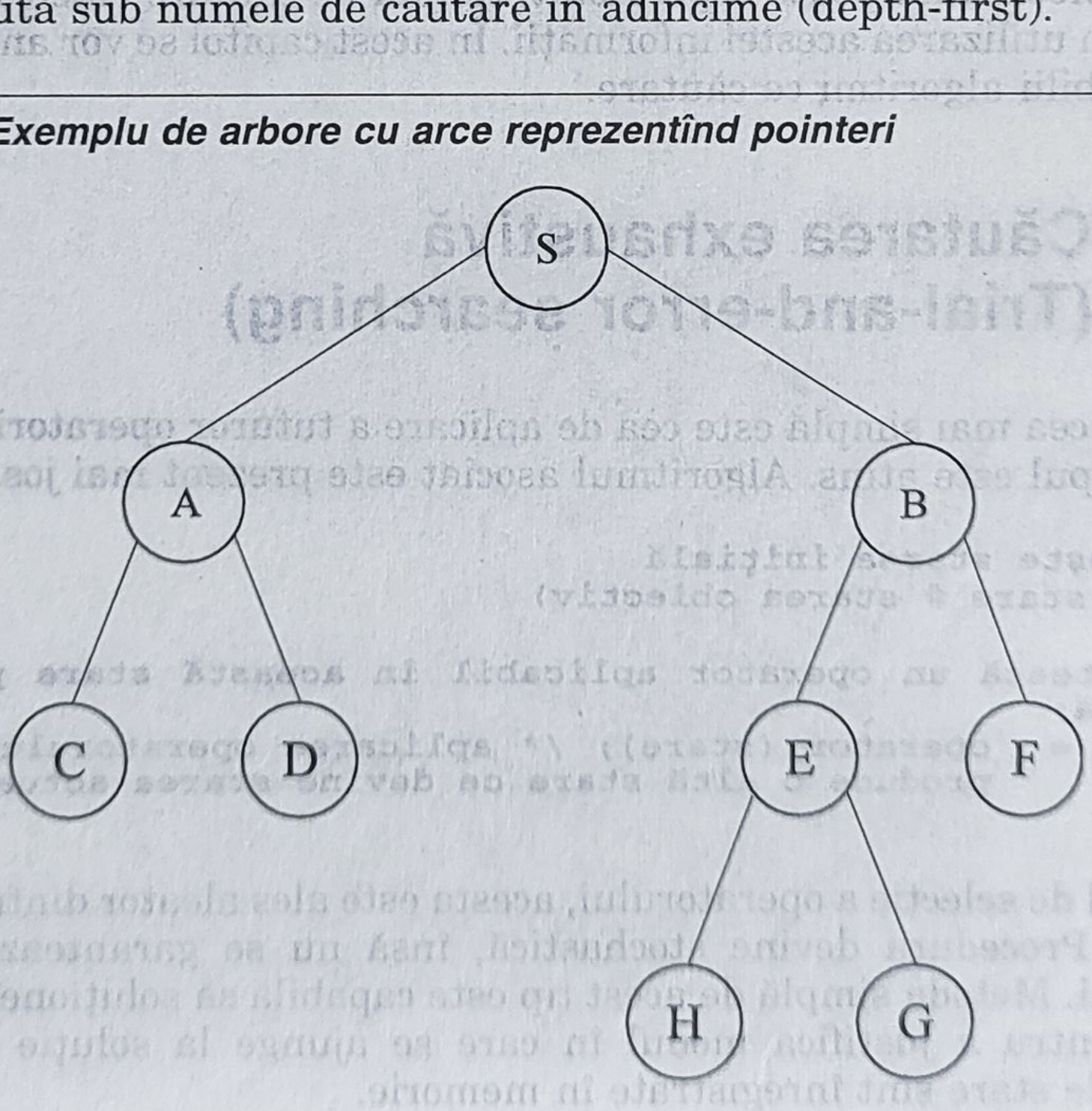
Această metodă permite examinarea spaţiului stărilor într-o secvenţă predeterminată, pînă cînd obiectivul este atins. Metoda poate fi segmentată în căutare pe verticală şi căutare pe orizontală, funcţie de ordinea în care procesul avansează în spaţiul stărilor.

6.2.1. Căutarea în adîncime

Căutarea în adîncime poate fi implementată pe calculator în mod direct prin utilizarea funcţiilor de înaintare şi revenire (*f, b*). Acest tip de căutare este caracterizat prin faptul că porneşte de la rădăcină şi traversează graful de la un nivel la altul prin urmărirea arcelor. O secvenţă a ramurii distincte $x_1, x_2,..., x_n$ este explorată cînd $b(x_1) = p$ (nod rărăcină), $b(x_2) = f(x_1),..., b(x_n) = f(x_{n-1})$ şi nu există x_{n+1} în L astfel încît $b(x_{n+1}) = f(x_n)$, ceea ce arată că $f(x_n)$ este o frunză. Cînd nodul dorit nu este găsit o altă secvenţă de ramificaţii către noduri frunză sînt iniţiate. Procesul continuă pînă cînd nodul dorit este localizat sau s-a ajuns la ultimul nod frunză.

Se consideră mai întîi căutarea cînd spaţiul stărilor problemei este reprezentat printr-un arbore ca cel din fig. 6.1. Metoda de căutare verticală este cunoscută sub numele de căutare în adîncime (depth-first).

▼ ***Fig. 6.1. Exemplu de arbore cu arce reprezentînd pointeri***



Căutarea porneşte de la nodul *S* şi continuă pînă cînd nodul obiectiv este atins. Dacă obiectivul este marcat prin *G* secvenţa de parcurgere este {*A, C, D, B, E, H, G*}. Dacă nodul *n* la care s-a ajuns nu este nodul obiectiv (în cazul de faţă *G*), şi dacă nodul *n* are noduri fiu, acestea vor fi examinate mai întîi. Dacă nodul curent nu are noduri fiu, programul returnează controlul la nodul părinte *m* al lui *n*, şi examinează următorul fiu ce porneşte din *m*. Calea către nodul obiectiv poate fi obţinută printr-o aranjare astfel încît nodul părinte al fiecărui nod fiu să fie reţinut în vederea căutărilor ulterioare. Modalitatea de realizare presupune asocierea la fiecare nod a unui pointer către nodul său părinte. În figură pointerii sînt indicaţi prin săgeţi. Cînd obiectivul este găsit calea de atingere a acestuia este determinată prin parcurgerea înapoi ghidată de secvenţa pointerilor.

Se dă mai jos algoritmul pentru căutarea nodurilor fiu ale nodului *n*, aplicînd toţi operatorii valizi ce expandează nodul *n*. Normal, în rezolvarea problemei pentru generarea arborilor de căutare sînt explorate procedeele de expandare a nodurilor. Arborele este definit indirect prin reprezentarea spaţiului stărilor. Nodurile ce au fost atinse dar nu au fost expandate sînt introduse într-o listă numită şi "open_list".

```
stabileşte nodul de start din listă;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
  else
  {
    n = prim_nod (lista); /* aduce primul nod din lista */
  }
  if obiectiv (n)) exit (succes); /* dacă n este obiectivul
  căutarea se termină cu succes */
  else
  {
    remove  (n, open_list); /* nodul n este scos din open_list */
          expandează n, pune toate nodurile fiu în open_list şi
          ataşează pointeri de la nodurile fiu la nodurile părinte
          */
  }
}
```

Obiectiv (*n*) este o funcţie logică ce întoarce valoarea TRUE dacă *n* este nod obiectiv şi FALSE altfel. Dacă *n* nu are noduri fiu procesul continuă cu următorul nod din listă. Nodurile fiu sînt introduse în listă în orice ordine, însă este mult mai eficient ca nodul obiectiv să fe primul în listă faţă de alte noduri ce au acelaşi părinte. Pentru arborele din figură succesiunea nodurilor în listă este următoarea:

$$(S) \rightarrow (A, B) \rightarrow (C, D, B) \rightarrow (D, B) \rightarrow (B) \rightarrow (E, F) \rightarrow (H, G, F) \rightarrow (G, H)$$

Pe parcursul parcurgerii grafului orice nod ce a fost expandat va fi extras din listă. În scopul urmăririi raţionamentului se pot dispune nodurile deja examinate într-o listă numită şi closed_list, care la început este vidă. Dacă se utilizează şi o astfel de listă algoritmul de lucru se modifică, structura sa fiind cea de mai jos.

```
stabileşte nodul de start în listă;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
  else
  {
    n = prim_nod (lista); /* aduce primul nod din listă */
  }
  if (obiectiv (n)) exit (succes); /* dacă n este obiectivul
     căutarea se termină cu succes */
  else
  {
    remove  (n, open_list); /* nodul n este scos din open_list */
    add  (n, closed_list); /* nodul n este adăugat în closed_list
         */
    expandează n, pentru a genera toate nodurile fiu. */ Orice nod
    ce apare în closed_list sau open_list va conţine şi pointeri
    la nodul sau părinte */
  }
}
```

La această procedură nodurile ce sînt adăugate în liste, prin expandare la open_list sau abandon la closed_list, vor fi plasate în vîrful stivei încît la următorul pas sînt primele luate în consideraţie.

6.2.2. Căutarea pe orizontală (în lărgime)

La căutarea pe orizontală (breadth-first search), nodurile de la acelaşi nivel sînt parcurse mai întîi. În exemplul din fig. 6.1 ordinea de căutare va fi *A, B, C, D, E, F, H, G*. Nodurile sînt examinate în ordine începînd cu nodul de start după algoritmul

```
pune nodul de start în open_list;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
  n = prim_nod (open_list);
  if (obiectiv (n)) exit (succes);
  remove (n, open_list);
  add (n, closed_list);
  expandează n, şi generează toate nodurile fiu. Acelea dintre
  noduri care nu sînt în open_list sau closed_list vor fi puse în
  partea inferioară a open_list cu pointeri asociaţi la nodul n;
}
```

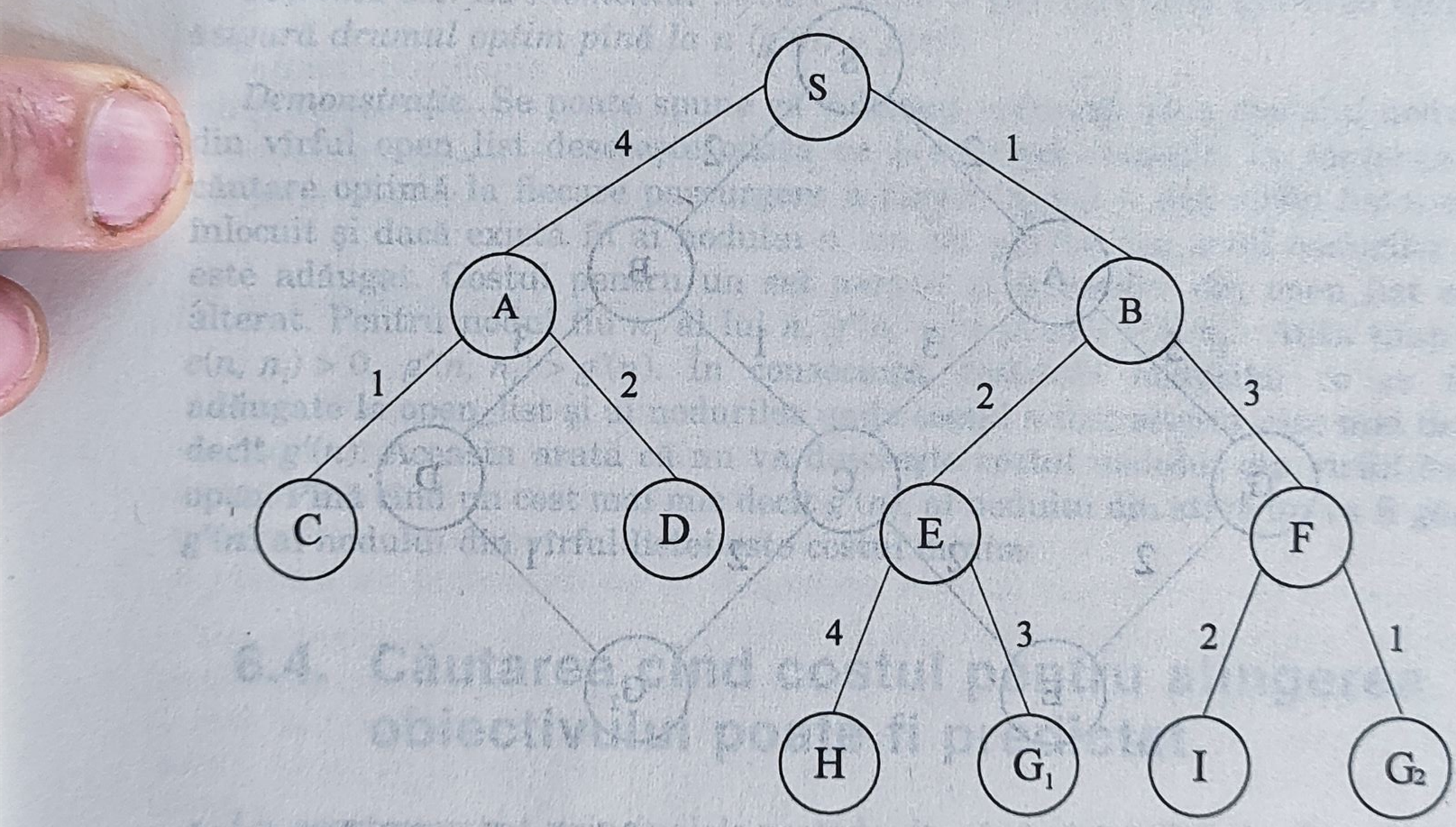
Procedura este similară cu cea de căutare în adîncime, excepţie făcînd faptul că nodurile generate sînt adăugate la capătul inferior al listei şi nu la cel superior. În general, dacă nodul obiectiv este în adîncimea arborelui, căutarea în adîncime este cea indicată, altfel se preferă căutarea în lărgime. Totuşi, dacă adîncimea arborelui şi numărul de ramuri ce pornesc de la un singur nod este mare se poate lua greu o decizie asupra metodei de căutare.

6.3. Căutarea soluţiei optime

La generarea spaţiului stărilor este posibil a fi aplicaţi acei operatori ce au condiţiile de aplicabilitate îndeplinite. Este normal ca eficientizarea căutării să fie făcută prin asocierea la fiecare operator a unei *funcţii de valoare, funcţii de cost* sau *funcţii de distanţă* ce oferă o măsură a consecinţelor aplicării operatorului.

Se consideră problema găsirii costului minim în cazul în care spaţiul stărilor este reprezentat printr-un graf şi costul operatorilor corespunzători fiecărui arc este dat. Se presupune că toate costurile sînt pozitive. Fie ca exemplu arborele din fig. 6.2 în care nodurile obiectiv sînt G_1 şi G_2.

▼ ***Fig. 6.2.*** *Arbore avînd costuri de traversare definite*

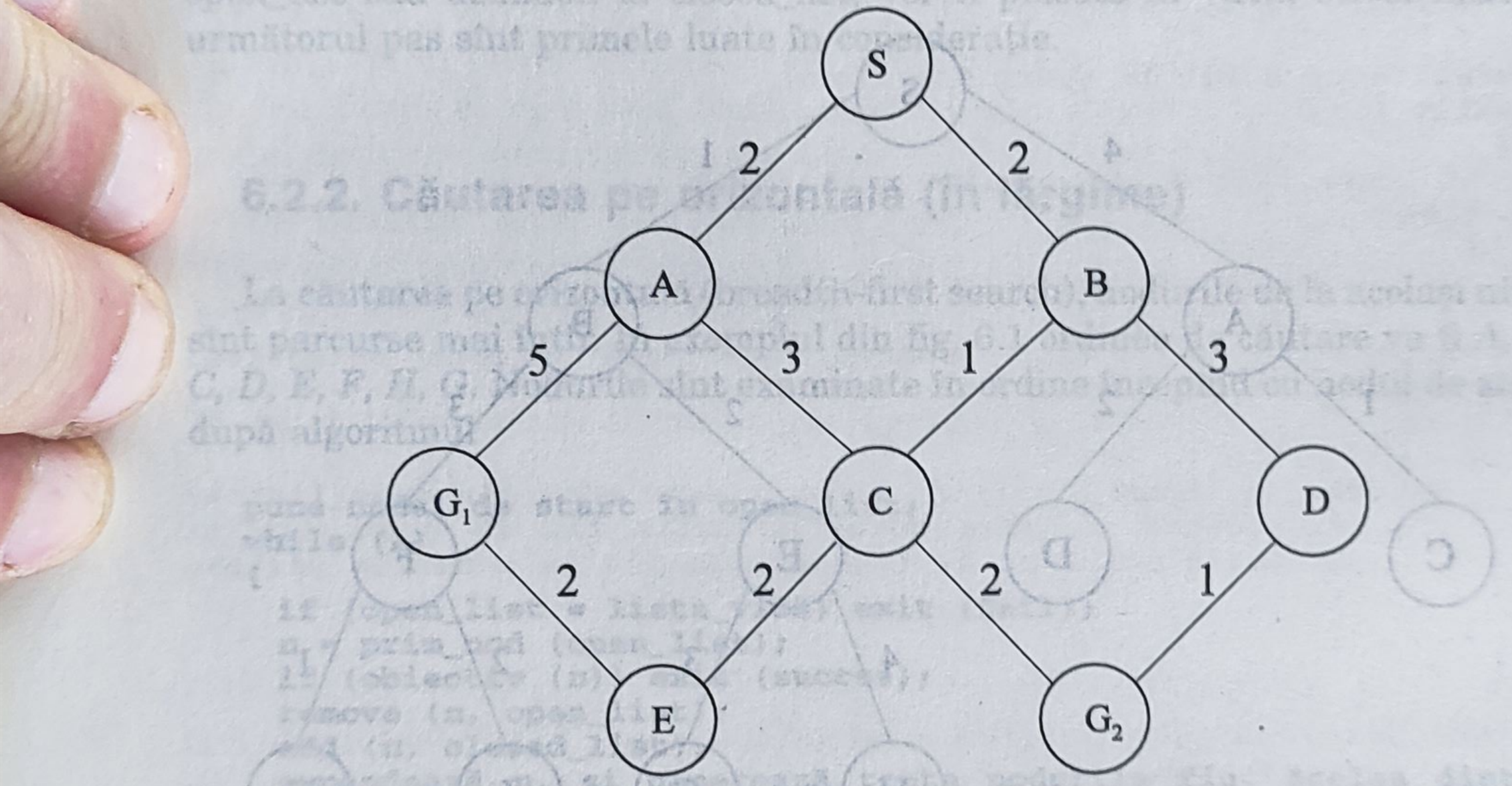


Strategia de căutare este bazată pe examinarea mai întîi a acelor noduri care au un cost minim pornind de la nodul de start. În exemplul de mai sus nodurile A şi B sînt generate prin expandarea nodului de start S. Întrucît atingerea nodului B implică cost minim pornind din S, se va examina B, care nu este nod obiectiv, avînd ca efect expandarea imediată în E şi F. Cum atingerea lui E implică un cost mai mic, acesta este următorul nod examinat, expandarea sa ducînd la nodul G_1.

Se va reprezenta costul drumului optim de la nodul de start la nodul n prin $g(n)$, costul celei mai bune parcurgeri prin $g'(n)$, adică $g'(n)$ este valoarea inferenţială a lui $g(n)$. Pentru o cale selectată $g(n) \geq g'(n)$. Algoritmul de căutare în arbori este:

```
pune nodul de plecare în open_list g'(n) = 0;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
  n = prim_nod (open_list);
  if (obiectiv (n)) exit (succes);
  remove (n, open_list);
  expandează n, calculează g'(n) pentru nodurile fiu ale lui n,
  ataşează pointeri de la n_i la n. Depune toate nodurile fiu în
  open_list şi aranjează nodurile în această listă în ordinea
  costului minim.
}
```

▼ *Fig. 6.3. Graph în care costul este definit*



Aplicarea algoritmului de mai sus pentru graful din fig. 6.3 determină succesiunea de noduri:

$$S(0) \rightarrow (A(2)B(3)) \rightarrow (B(3)C(5)G_1(7)) \rightarrow (C(4)D(6)G_1(7)) \rightarrow (G_2(6)D(6)E(6)G_1(7))$$

Se va arăta faptul că dacă graful este finit există un drum optim de la nodul de start la nodul obiectiv, prin aplicarea următoarelor teoreme. Soluţia se obţine prin procedura optimală de căutare.

Teorema 6.1. Într-un graf finit există un optim de la nodul de start la nodul obiectiv pe care procedura îl găseşte.

Demonstraţie. Se va presupune că algoritmul de căutare nu se termină cu succes. În acest caz open_list este vidă, closed_list conţine toate nodurile explorate pînă în acel moment. Deoarece graful are o soluţie se va presupune că drumul conţie nodurile (n_0, n_1,..., n_m), în care n_0 este identic cu nodul S şi n_m este identic cu nodul obiectiv. Parcurgînd drumul indicat înapoi se găseşte nodul n_i care a fost introdus în closed_list iar nodul n_{i+1} nu se găseşte în closed_list. Întrucît n_i este inclus în closed_list el va fi expandat. Ca urmare a expandării n_{i+1} va fi pus mai întîi în open_list. Ca urmare n_{i+1} trebuie să fie examinat înainte ca open_list să devină vidă. Dacă n_{i+1} este un nod obiectiv căutarea duce la succes. Dacă n_{i+1} nu este un nod obiectiv este pus în closed_list. În ambele cazuri presupunerea de eşec este contrazisă. Întrucît graful este finit şi în urma căutării nu se ajunge la eşec atunci trebuie să se obţină un succes.

Raţiunea pentru care un nod fiu n_i ce se găseşte în closed_list nu este şi în open_list porneşte de la faptul că $g'(n, n_i)$ nu poate fi mai mic decît $g'(n_i)$, aşa că o cale trecînd prin n nu poate fi considerată un candidat la soluţie, aşa cum se demonstrează în teorema următoare.

Teorema 6.2. La momentul în care nodul n este expandat căutarea optimă asigură drumul optim pînă la n ($g'(n) = g(n)$).

Demonstraţie. Se poate spune că valoarea inferenţiată a costului nodului din vîrful open_list descreşte odată cu avansarea căutării. În secvenţa de căutare optimă la fiecare parcurgere a buclei, nodul n din vîrful listei este înlocuit şi dacă există fii ai nodului n, un set parţial din setul nodurilor fiu este adăugat. Costul pentru un set parţial al nodurilor din open_list este alterat. Pentru nodul fiu n_i al lui n, $g'(n, n_i) = g'(n) + c(n, n_i)$. Atîta timp cît $c(n, n_i) > 0$, $g'(n, n_i) > g'(n)$. În consecinţă, costurile nodurilor ce au fost adăugate la open_list şi al nodurilor unde costul a fost alterat este mai mare decît $g'(n)$. Aceasta arată că nu va descreşte costul nodului din vîrful listei open. Pînă cînd un cost mai mic decît $g'(n)$, al nodului din stivă nu va fi găsit, $g'(n)$ al nodului din vîrful listei este costul minim.

6.4. Căutarea cînd costul pentru atingerea obiectivului poate fi predictat

La acest moment principalele metode de căutare grafică au fost expuse. Cum problema are o reprezentare în spaţiul stărilor, căutarea poate fi făcută mai eficient utilizînd informaţiile pentru reprezentarea problemei. De exemplu, în problema labirintului dacă se cunoaşte poziţia obiectivului relativ la punctul de plecare, se va selecta calea ce încarcă cel mai puţin posibil costul. La problema careului cu opt numere, o informaţie a apropierii de obiectiv este dată de poziţia punctelor faţă de starea obiectiv. Informaţia rezultată din predicţia costului faţă de obiectiv, chiar dacă nu este suficient de precisă poate fi utilizată. Buclele moarte pot determina creşterea timpului (şi implicit

a costului) de atingerea obiectivului. Astfel de cunoştinţe sînt de natură euristică, întrucît nu garantează corectitudinea complectă. Căutarea bazată pe cunoşinţe euristice se numeşte şi căutare euristică. Se cunosc mai multe tehnici de căutare euristică, tehnici ce se vor expune mai jos.

6.4.1. Metoda gradientului (Hill-climbing method)

Ideea de bază a căutării în cazul unei strategii irevocabile este cea prin care pornind de la starea iniţială să se minimizeze distanţa ce separă starea curentă obţinută, de starea obiectiv a problemei. Metodele matematice cunoscute în teoria sistemelor găsesc un bun teren de aplicabilitate în această situaţie. O metodă des utilizată este metoda gradientului (metoda hill climbing) ce se utilizează în procesele de căutare multidimensională a extremelor funcţiilor de mai multe variabile sau a soluţiilor sistemelor de ecuaţii neliniare. O serie de probleme duc la complexitate în procesul generării stărilor ce se reflectă în caracterul neterminal al stărilor obţinute. Aceasta nu permite testarea concordanţei cu obiectivul pentru toate stările posibil de obţinut dintr-un nod dat. Generarea spaţiului stărilor se desfăşoară după strategii ce urmăresc două criterii de optimalitate:

- *optimalitate locală* - avînd ca obiectiv găsirea numărului minim de paşi ce duce la o stare terminală, fără epuizarea tuturor combinaţiilor posibile. Starea precedentă este marcată pentru a se asigura revenirea ulterioară în caz de eşec şi reluarea procesului prin generarea unui nou element terminal;
- *optimalitatea globală* - se referă la obţinerea soluţiei problemei cu un număr minim de reveniri.

Strategiile de control irevocabile se bazează pe cunoaşterea locală, determină operatorul aplicat de la un pas la altul, noua stare reprezintă un context necunoscut, context folosit pentru determinarea unui alt operator pînă se atinge obiectivul. Se poate spune că secvenţa de operatori defineşte o cunoaştere globală despre soluţia problemei ce se determină ca urmare a aplicării operatorilor prin strategia de control.

Ideea de bază a acestei metode se bazează pe faptul că direcţia celei mai mari creşteri a funcţiei $f(x_1, x_2, \ldots, x_n)$ este dată de gradientului funcţiei.

$$\text{grad}\, f(x_1, x_2, \ldots, x_n) = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

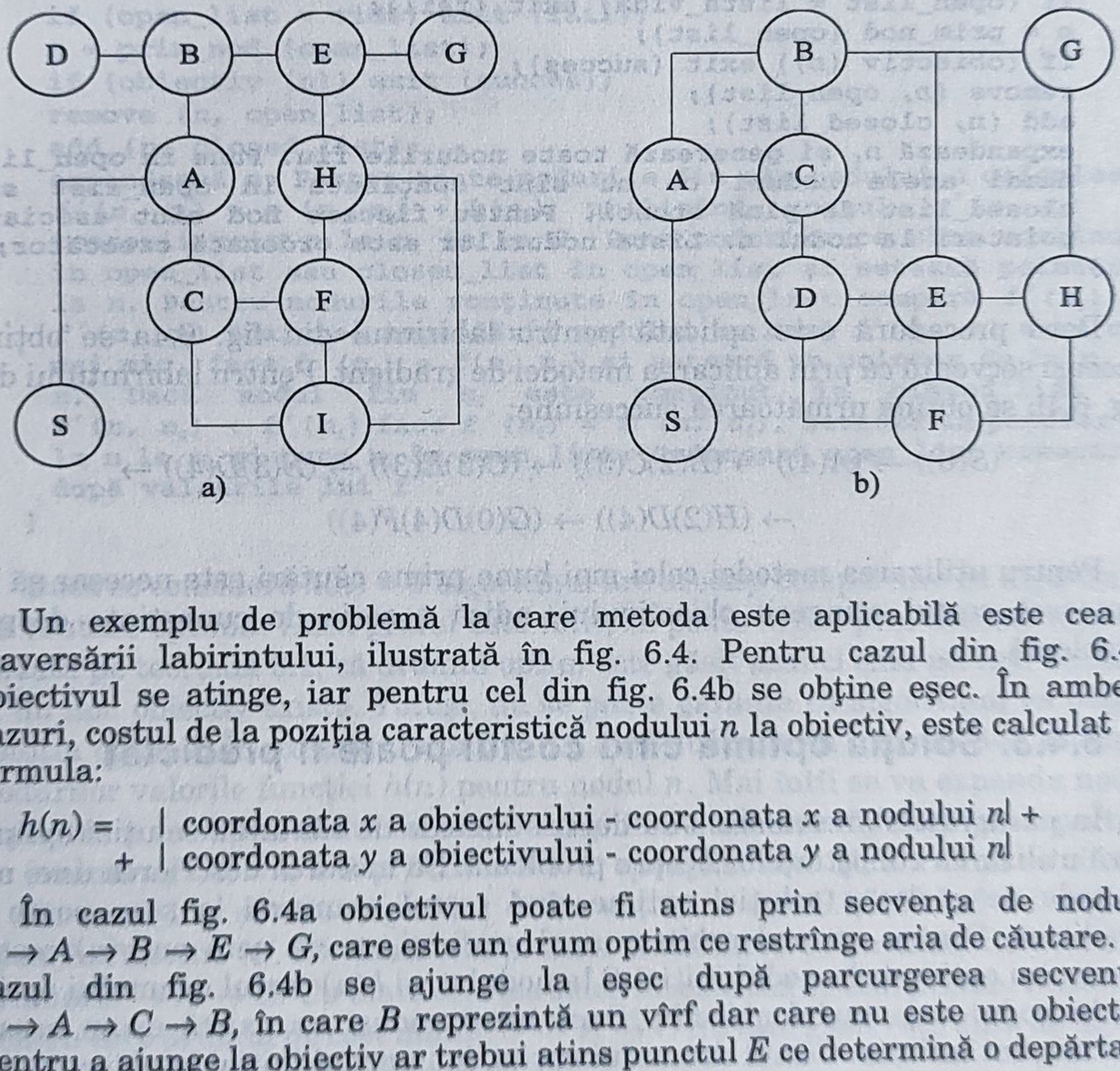
Aplicarea metodei este dificilă dacă numărul variabilelor este mare, numărul funcţiilor de cost este mare, funcţia are mai multe valori extreme.

La căutarea într-un graf, strategia traversării pentru atingerea obiectivului prin alegerea acelor noduri care sînt predictate a fi apropiate de obiectiv este numită metoda de gradient. Specific pentru alegerea următorului nod după n, se calculează $h(n_i)$ pentru toate nodurile fiu $(n_1, n_2, \ldots, n_m)$ ale lui n şi stabileşte ca următor nod acel nod pentru care $h(n_i)$ este mai mic. Funcţia $h(n)$ este o măsură a apropierii de obiectiv. Algoritmul de căutare este:

```
n = nodul de start;
while (1)
{
  if (obiectiv (n)) exit (succes);
  expandează n, calculează h(n_i) pentru toate nodurile fiu n_i şi
  face nodul fiu nod următor (next_n) acel nod care are valoarea
  minimă;
  if (h(n) < h(next_n)) exit (fail);
  n = next_n;
}
```

Se poate imagina obiectivul ca vîrful unui munte şi $h(n)$ ca diferenţă de înălţime între nodul n şi vîrf. Metoda urmăreşte apropierea de obiectiv prin gradient maxim. Dacă există numai un vîrf metoda permite atingerea obiectivului, însă pot exista vîrfuri locale ce sînt atinse, ce nu permit apoi apropierea de maximul global.

▼ ***Fig. 6.4. Exemple de căutare utilizînd metode de gradient***
a) caz în care se obţine succes
b) caz în care se ajunge la eşec



Un exemplu de problemă la care metoda este aplicabilă este cea a traversării labirintului, ilustrată în fig. 6.4. Pentru cazul din fig. 6.4a obiectivul se atinge, iar pentru cel din fig. 6.4b se obţine eşec. În ambele cazuri, costul de la poziţia caracteristică nodului n la obiectiv, este calculat cu formula:

$$h(n) = |\text{coordonata } x \text{ a obiectivului - coordonata } x \text{ a nodului } n| + |\text{coordonata } y \text{ a obiectivului - coordonata } y \text{ a nodului } n|$$

În cazul fig. 6.4a obiectivul poate fi atins prin secvenţa de noduri $S \rightarrow A \rightarrow B \rightarrow E \rightarrow G$, care este un drum optim ce restrînge aria de căutare. În cazul din fig. 6.4b se ajunge la eşec după parcurgerea secvenţei $S \rightarrow A \rightarrow C \rightarrow B$, în care B reprezintă un vîrf dar care nu este un obiectiv. Pentru a ajunge la obiectiv ar trebui atins punctul E ce determină o depărtare de obiectiv.

6.4.2. Metoda celei mai bune prime căutări

În metoda gradientului maxim dacă sînt şi alte extreme locale (valori minime ale lui $h(n)$) decît obiectivul nu se poate garanta continuarea algoritmului pentru atingerea acestuia. Tehnica cunoscută sub numele de metoda celei mai bune prime căutări se bazează pe luarea deciziei după examinarea spaţiului stărilor. Toate nodurile ce au fost examinate prin această tehnică pînă la momentul curent sînt stocate, urmînd ca nodul cel mai apropiat de obiectiv să fie expandat. Pînă aici metoda este similară cu cea bazată pe gradient, însă atunci cînd $h(n)$ devine constant sau acţiunile permise în starea curentă produc depărtarea de obiectiv, un alt nod din listă va fi expandat. Algoritmul pentru căutarea în grafuri după această metodă este dat mai jos.

```
pune nodul de start S în open_list;
while (1)
{
  if (open_list = lista_vidă) exit (fail);
  n = prim_nod (open_list);
  if (obiectiv (n)) exit (succes);
  remove (n, open_list);
  add (n, closed_list);
  expandează n, şi generează toate nodurile fiu. Pune în open_list
  numai acele noduri ce nu sînt conţinute în open_list sau
  closed_list de pînă atunci. Pentru fiecare nod sînt asociaţi
  pointeri la nodul n. Lista nodurilor este ordonată crescător;
}
```

Dacă procedura este aplicată pentru labirintul din fig. 6.4a se obţine aceeaşi secvenţă ca prin aplicarea metodei de gradient. Pentru labirintului din fig. 6.4b se obţine următoarea succesiune:

$$(S(6)) \rightarrow (A(4)) \rightarrow (B(2)C(3)) \rightarrow (C(3)E(3)) \rightarrow (E(3)D(4)) \rightarrow$$
$$\rightarrow (H(2)D(4)) \rightarrow (G(0)D(4)F(4))$$

Pentru utilizarea metodei celei mai bune prime căutări este necesar să se cunoască costul atingerii obiectivului, adică o serie de cunoştinţe despre problemă.

6.4.3. Soluţia optimă cînd costul poate fi predictat

În paragrafele anterioare s-au descris metode de căutarea soluţiei optime fără utilizarea cunoştinţelor despre problemă. Se încearcă descrierea unei noi metode de găsire a soluţiei optime cînd costul ajungerii la scop poate fi predictat. Pentru un nod arbitrar n al grafului, se va nota cu $g(n)$ costul drumului optim de la nodul iniţial S la nodul n, şi $h(n)$ costul drumului optim de la nodul n la nodul obiectiv. Dacă între noduri nu există drum aceste costuri sînt infinite. Costul drumului optim de la nodul iniţial la obiectiv este dat prin următoarea formulă:

$f(n) = g(n) + h(n)$

şi problema este de a găsi drumul optim (partea care dă $f(S)$) de la nodul de start. Dacă $f(n)$ este cunoscută exact, soluţia poate fi obţinută prin înlănţuirea nodurilor cu cel mai mic $f(n)$ ce pornesc de la S. În practică fie $g(n)$ fie $h(n)$ nu sînt cunoscute cu precizie, aşa că este necesară căutarea.

Ca şi în exemplele precedente, valoarea costului inferenţiat $h(n)$ către obiectiv este dat pentru toate nodurile. Se va nota cu $g'(n)$ valoarea lui $g(n)$ pentru nodurile deja generate ce au cost minim şi conţin un drum ce trece prin n. Valoarea inferenţiată $f'(n)$ a costului drumului optim trecînd prin n este dată de formula

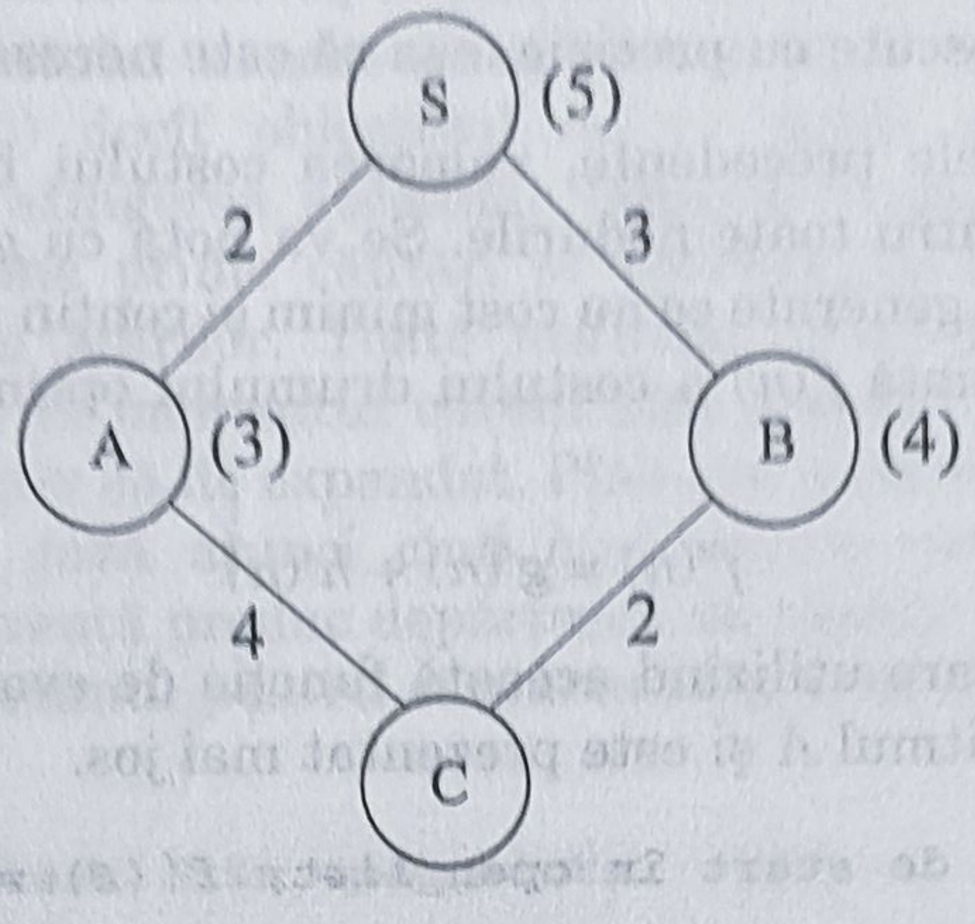
$$f'(n) = g'(n) + h'(n)$$

Strategia de căutare utilizînd această funcţie de evaluare este cunoscută sub numele de algoritmul A şi este prezentat mai jos.

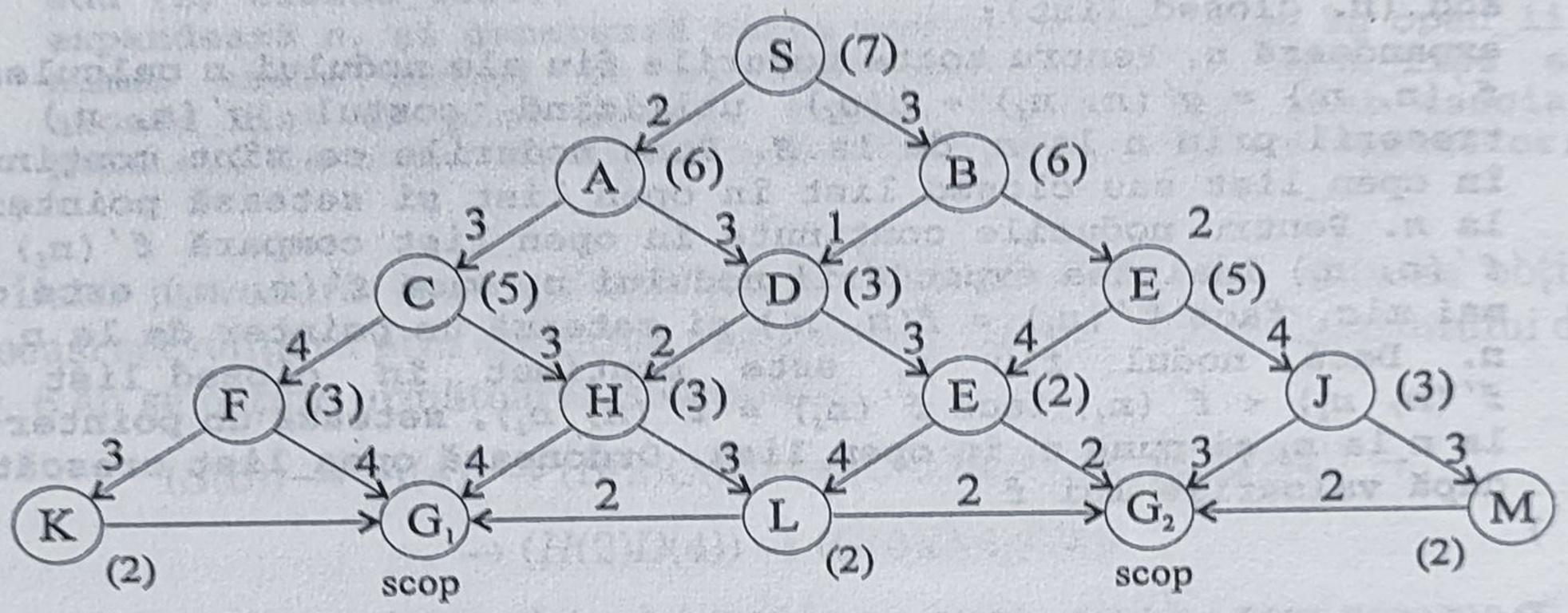
```
Introduce nodul de start în open_list; f'(S) = h'(S);
While (1)
{
  if (open_list = vidă) exit (fail);
  n = prim_nod (open_list);
  if (obiectiv (n)) exit (succes);
  remove (n, open_list);
  add (n, closed_list);
  expandează n. Pentru toate nodurile fiu ale nodului n calculează
  f'(n, n_i) = g'(n, n_i) + h(n_i)  utilizînd  costul  g'(n, n_i)  al
  trecerii prin n la n_i de la S. Pune nodurile ce sînt conţinute
  în open_list sau closed_list în open_list şi setează pointerii
  la n. Pentru nodurile conţinute în open_list compară f'(n_i) şi
  f'(n, n_i) înaintea expandării nodului n. Dacă f'(n, n_i) este cel
  mai mic, face f'(n_i) = f(n, n_i) şi setează un pointer de la n_i la
  n.  Dacă  nodul  fiu  n_i  este  conţinut  în  closed_list  şi
  f'(n, n_i) < f'(n_i) face f'(n_i) = f'(n, n_i), setează un pointer de
  la n la n_i şi pune n_i în open_list. Ordonează open_list crescător
  după valoarile lui f'.
}
```

Dacă se consideră $h(n) = 0$ algoritmul are aceeaşi comportare cu algoritmul de căutare optimă. Dacă graful este finit, se poate arăta prin aceeaşi metodă bazată pe teorema 6.1, că drumul optim este găsit atunci cînd un nod de start şi un nod obiectiv există. Totuşi, nu se poate garanta că algoritmul va obţine soluţia optimă. În exemplul din fig. 6.5 sînt ilustrate în parantezele asociate nodurilor valorile funcţiei $h(n)$ pentru nodul n. Mai întîi se va expanda nodul S pentru a se obţine nodurile sale fiu A şi B. Se calculează funcţiile respective de evaluare $f'(A) = 2 + 3$ şi $f'(B) = 3 + 4$, open_list este (A, B). Cînd se citeşte A din open_list şi este expandat se va obţine nodul G. $f'(G) = 6 + 0$ aşa că open_list devine (G, B). Cînd este preluat G căutarea se termină. Soluţia este drumul $S \rightarrow A \rightarrow G$. Cu toate acestea calea nu este cea de cost minim. Raţiunea pentru care drumul de cost minim $S \rightarrow B \rightarrow G$ nu este găsit este dată de faptul că $f(B)$ devine mai mare decît $f'(B)$.

▼ *Fig. 6.5. Graf în care algoritmul A* nu găseşte soluţia optimă*



▼ *Fig. 6.6. Graf în care condiţia h'(n) <= h(n) este îndeplinită*



Se poate dovedi că dacă $h'(n)$ excede $h(n)$, algoritmul va găsi soluţia optimă. Dacă costul inferenţiat este marginea inferioară a costului real ($h'(n) <= h(n)$) algoritmul A se transformă în algoritmul $A*$.

În continuare se dă un nou exemplu de căutare într-un graf utilizînd algoritmul $A*$. Graful în care se execută căutarea este prezentat în fig. 6.6. Ca şi în fig. 6.5 valoarea lui $h'(n)$ este dată în paranteze asociate nodului. Schimbările în open_list obţinute prin utilizarea algoritmului $A*$ sînt marcate de indicarea lui $f'(n)$ în parantezele asociate nodurilor.

Se va obţine secvenţa de noduri în open_list:

$$(S(7)) \rightarrow (A(8)B(9)) \rightarrow (D(8)B(9)C(10)) \rightarrow (B(9)C(10)H(10)I(10))$$
$$\rightarrow (D(7)C(10)E(10)H(10)I(10)) \rightarrow (H(9)I(9)C(10)E(10))$$
$$\rightarrow (I(9)G_1(10)C(10)E(10)L(11)) \rightarrow (G_2(9)G_1(10)C(10)E(10)L(11))$$

Procedura se termină cînd G_2 este extras din open_list. Soluţia $S \to B \to D \to I \to G_2$ este soluţia optimă ($h(S) = 9$). Este remarcabil faptul că un nod n care satisface $h'(n) <= h(S)$ trebuie să existe în open_list şi cel puţin pentru nodurile din calea optimă $h'(n) <= h(S)$. În procesul de căutare nodul D este introdus în closed_list, în schimb atunci cînd se reîntoarce la D şi găseşte drumul de cost minim acesta este pus din nou în open_list. Pentru că $h' = 0$ este sigur marginea inferioară pentru h, căutarea optimă utilizînd această procedură este un caz particular al algoritmului A^*. Algoritmul A^* corespunde căutării optimale fără utilizarea cunoştinţelor euristice.

6.4.4. Caracteristici ale algoritmului A^*

Întrucît algoritmul A^* este un caz particular al algoritmului A, posedă toate caracteristicile acestuia din urmă. Dacă graful este finit şi există un drum de la nodul de start la nodul obiectiv, algoritmul se termină prin găsirea unui drum între nodul de start şi nodul obiectiv. Se arată mai jos că şi în cazul în care graful este infinit algoritmul găseşte soluţia optimă.

Lema 6.1. *Reprezentînd nodul de start prin S, înaintea terminării algoritmului A*, va putea exista un nod n care satisface relaţia $f'(n) <= f(S)$.*

Demonstraţie. Drumul optim de la S la nodul obiectiv este reprezentat printr-o serie $(n_0, n_1, ..., n_m)$ de noduri ($n_0 = S$, n_m = nod obiectiv). Iniţial S se găseşte în open_list şi algoritmul nu s-a terminat încă, aşa că este un nod al drumului optim dispus în open_list. Referitor la nodurile din open_list, notînd cu n nodul care apare primul în secvenţa nodurilor drumului optim, nodurile antecedente acestuia sînt conţinute în closed_list, aşa că drumul optim de la nodurile părinte lui n a fost găsit. Întrucît n aparţine drumului optim de la S la obiectiv, drumul optim ce trece prin n a fost găsit. De aceea

$$f'(n) = g'(n) + h'(n) = g(n) + h'(n)$$

Întrucît în algoritmul A^* $h'(n) <= h(n)$

$$f'(n) <= g(n) + h(n) = f(n)$$

şi deci n aparţine drumului optim, aşa că $f'(n) <= f(S)$. De aici rezultă că există un nod n care satisface $f(n) = f(S)$.

Lema 6.2. *Dacă graful conţine un drum de la nodul de start la nodul obiectiv, drumul către nodul obiectiv poate fi găsit prin algoritmul A*.*

Demonstraţie. S-a arătat pînă în prezent că dacă algoritmul A^* se termină, el poate găsi calea către nodul obiectiv. Se presupune că algoritmul A^* nu se termină şi că lungimea arcului este reprezentată prin 1. Valoarea minimă a lungimii drumului de la nodul de start la nodul n este notată cu $d(n)$. Cum numărul ramurilor de la un singur nod este finit, continuarea căutării determină $d(n)$ pentru orice nod n din open_list ce devine mai mare decît un număr arbitrar M. Întrucît costul este pozitiv făcînd valoarea minimă m se obţine $g(n) >= d(n)m$. Întrucît, $g'(n) >= g(n)$ şi $f''(n) >= g'(n)$,

$f'(n) >= d(n)m$. Dacă se alege $M = f(S)/m$, căutarea produce $d(n)$ ce devine mai mare decît M, aşa că se obţine $f'(n) >= f(S)d(n)/M > f(S)$. Aceasta contrazice Lema 6.1, deci algoritmul se termină.

Teorema 6.3. *Dacă graful conţine un drum de la nodul de start la nodul obiectiv algoritmul A* se termină prin găsirea soluţiei optime.*

Demonstraţie. Se cunoaşte din lema 6.2 ca algoritmul A^* se termină. S-a arătat de asemenea că dacă algoritmul se termină, drumul spre nodul obiectiv este găsit, precum şi faptul că terminarea algoritmului cu găsirea nodului obiectiv asigură soluţia optimă. Dacă se presupune că aceasta nu este soluţia optimă, atunci $f(G) > f(S)$. Din lema 6.1, înainte de terminarea căutării există un nod n în open_list care satisface $f'(n) <= f(S)$. Întrucît pentru nodul n, $f'(n) < f'(G)$ la acest moment, algoritmul A^* va selecta n şi nu G. Aceasta contrazice presupunerea că se termină căutarea prin găsirea lui G.

Corolar 6.1. *$F'(n')$ pentru nodul n' expandat prin algoritmul A^* este mai mare decît costul soluţiei optime.*

Demonstraţie. Nodul n' ce este expandat nu este un nod obiectiv şi căutarea nu s-a terminat. Din lema 6.1 open_list conţine un nod n care satisface $f'(n) <= f(S)$. Algoritmul preia nodul pentru care f este minim din open_list şi îl expandează ($f'(n') <= f'(n)$). De aceea $f'(n') <= f(S)$. Din teorema 6.3 se poate vedea că atunci $h'(n) = 0$, aşa că algoritmul de căutare optimă permite găsirea soluţiei optime.

Se poate acum examina relaţia între funcţia euristică $f'(n)$ şi algoritmul A^*. Dacă $f'(n) = f(n)$, soluţia poate fi găsită fără expandarea nodurilor ce nu sînt necesare. Se poate spera că apropierea lui $f(n)$ de $f'(n)$ va scădea numărul nodurilor ce sînt expandate. Presupunînd că se utilizează algoritmul A^* cu două funcţii diferite de evaluare $f'_1(n)$ şi $f'_2(n)$ (A_1 şi A_2), dacă pentru toate nodurile n, $h_1(n) > h_2(n)$, se spune că gradul de cunoaştere pentru A_1 este mai mare decît cel pentru A_2.

Teorema 6.4. *Fie doi algoritmi A^*, A_1 şi A_2, presupunînd că gradul de cunoaştere pentru A_1 este mai mare decît cel pentru A_2 nodurile ce sînt expandate prin A_1 vor putea fi expandate şi prin A_2.*

Demonstraţie. Se va utiliza pentru demonstraţie metoda inducţiei matematice. Teorema reţine punctul final al căii de lungime 0 de la nodul de start. (Dacă S este un nod obiectiv nici A_1, nici A_2 nu va determina expandarea, altfel ambele sînt expandate). Se va arăta că dacă teorema reţine nodul pentru punctul de terminare al tuturor drumurilor de lungime k de la S, reţine de asemenea toate punctele de terminare ale căilor de lungime k - 1. Se presupune că acesta este un nod expandat prin A_1 dar neexpandat prin A_2. Denumind acest nod n, întrucît A_2 expandează nodul părinte, cînd A_2 se termină, n este scos din open_list. Cum n nu este selectat, cînd algoritmul se termină, se reţin următoarele relaţii:

$f'(n) >= f(S)$

Însă $f'(n) = g'_2(n) + h'_2(n)$,

$h'_2(n) >= f(S) - g'_2(n)$.

Pe de altă parte, întrucît A_1 a expandat n la acest moment

$f'_1(n) <= f(S)$, şi

$f'_1(n) = g'_1(n) + h'_1(n)$ se obţine

$h'_1(n) <= f(S) - g'_1(n)$.

Întrucît orice nod care este expandat prin A_1 ce este nod în calea de lungime k va fi expandat şi prin A_2:

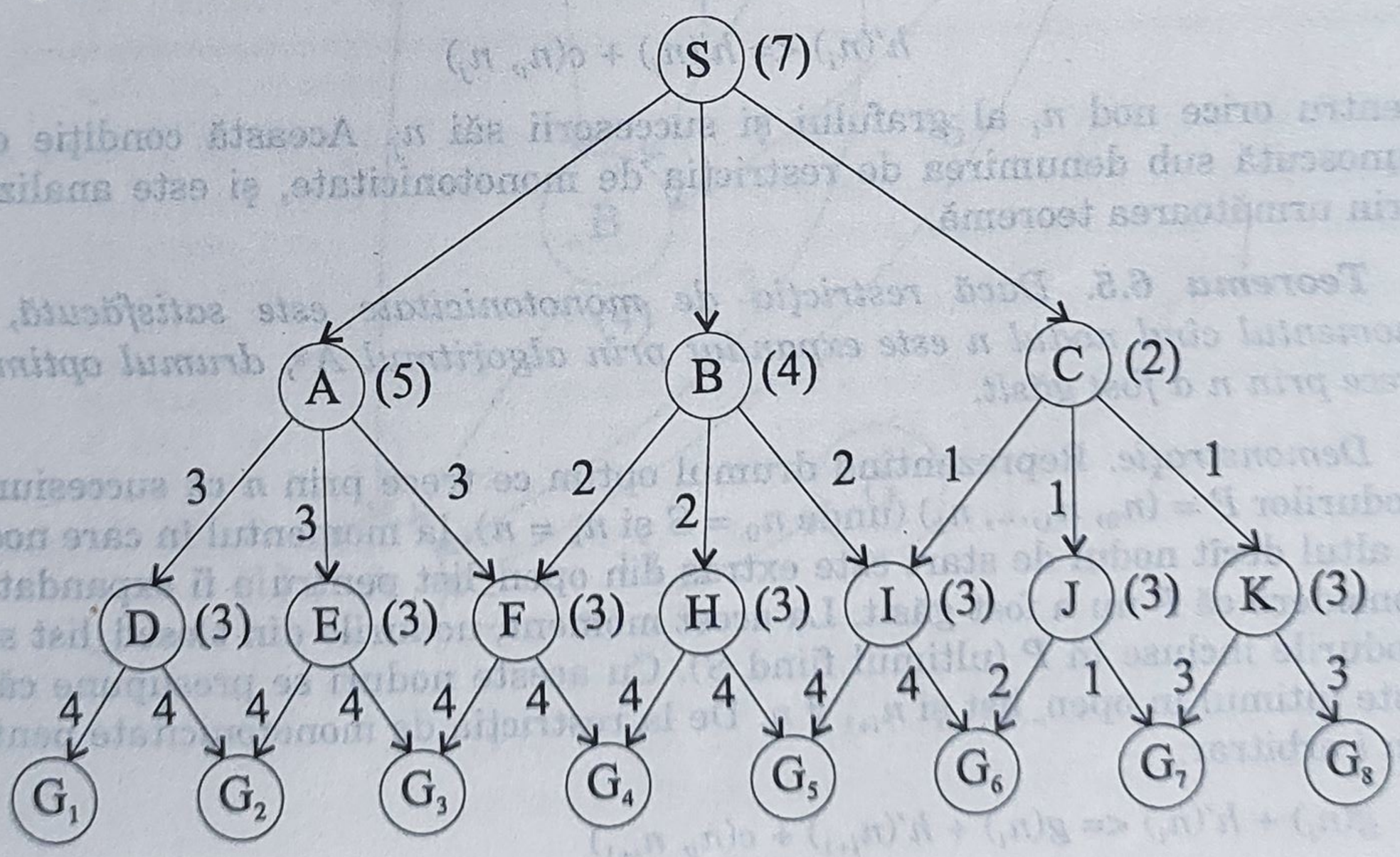
$g'_2(n) <= g'_1(n)$, şi

$h'_1(n) <= f(S) - g'_1(n) <= f(S) - g'_2(n) <= h'_2(n)$.

Aceasta contravine presupunerii că $h'_1 > h'_2$. Deci presupunerea că n există nu este adevărată.

De la această teoremă se poate vedea că algoritmul A^* utilizînd cunoaşterea euristică $h(n)$ expandează mai puţine noduri decît căutarea pentru soluţia optimă fără utilizarea lui h' ($h' = 0$). În fig. 6.7 se indică un exemplul al utilizării algoritmului. Indiferent de tipul algoritmului de căutare se găseşte soluţia optimă ($S\ C\ J\ G_7$) avînd costul asociat 8. Atunci cînd $h' = 0$ toate nodurile către obiectivele G_1 la G_8 sînt expandate. Aceasta corespunde găsirii valorii minime a costurilor după găsirea tuturor drumurilor la obiectiv. În contrast, dacă valorile indicate în paranteze în fig. 6.7 sînt utilizate pentru funcţia h, nu vor fi expandate decît acele noduri ce conduc la soluţia optimă.

▼ ***Fig. 6.7. Exemplu indicînd efectul funcţiei euristice h( )***



6.4.5. Îmbunătăţiri ale algoritmului *A**

Numărul de noduri expandate este o măsură a eficienţei de căutare. Aşa cum se observă din algoritmul prezentat, nodurile ce au fost expandate sînt trecute în closed_list şi pot fi readuse în open_list. Altfel spus, este posibil ca acelaşi nod să fie expandat de mai multe ori, la momente de timp diferite. Cu cît numărul de noduri ce se expandează scade, eficienţa căutării creşte. În fig. 6.8 se arată un exemplu de graf în care algoritmul *A** expandează acelaşi nod de mai multe ori.

Operaţiile de expandare a nodurilor utilizînd acest algoritm sînt arătate în tabela 6.1. Fiecare rînd din tabelă indică nodurile din open_list şi closed_list, precum şi costul inferenţiat corespunzător f'. Pentru a indica faptul că un nod este în closed_list în tabelă s-a marcat prin simbolul *. Iniţial, numai nodul de start S este în open_list, aşa că $f'(S)$ este dată în primul rînd al tabelei.

Apoi, S este pus în closed_list şi nodurile A, B, C şi D care sînt obţinute prin expandarea lui S sînt dispuse în open_list. În continuare, nodul A pentru care f' este un minim este extras din open_list şi depus în closed_list odată cu fiii săi. Cînd procedura este repetată nodul se va expanda în total de 17 ori. Dacă numărul de noduri este N, se cunoaşte că numărul maxim de expandări ale nodurilor este 2^N. În opoziţie, dacă nu se utilizează cunoaşterea euristică toate nodurile vor fi expandate numai o singură dată. Pentru ca algoritmul *A** să nu expandeze acelaşi nod de două ori este nevoie să se impună următoarea condiţie

$$h'(n_i) <= h'(n_j) + c(n_i, n_j)$$

pentru orice nod n_i al grafului şi succesorii săi n_j. Această condiţie este cunoscută sub denumirea de restricţia de monotonicitate, şi este analizată prin următoarea teoremă.

Teorema 6.5. *Dacă restricţia de monotonicitate este satisfăcută, la momentul cînd nodul n este expandat prin algoritmul A*, drumul optim ce trece prin n a fost găsit.*

Demonstraţie. Reprezentînd drumul optim ce trece prin n ca succesiunea nodurilor $P = (n_0, n_1,..., n_k)$ (unde $n_0 = S$ şi $n_k = n$), la momentul în care nodul n altul decît nodul de start este extras din open_list pentru a fi expandat se consideră că P nu a fost găsit. La acest moment, nodurile din closed_list sînt nodurile incluse în P (ultimul fiind S). Cu aceste noduri se presupune că n_i este ultimul în open_list şi n_{i+1} # n. De la restricţia de monotonicitate pentru un i arbitrar:

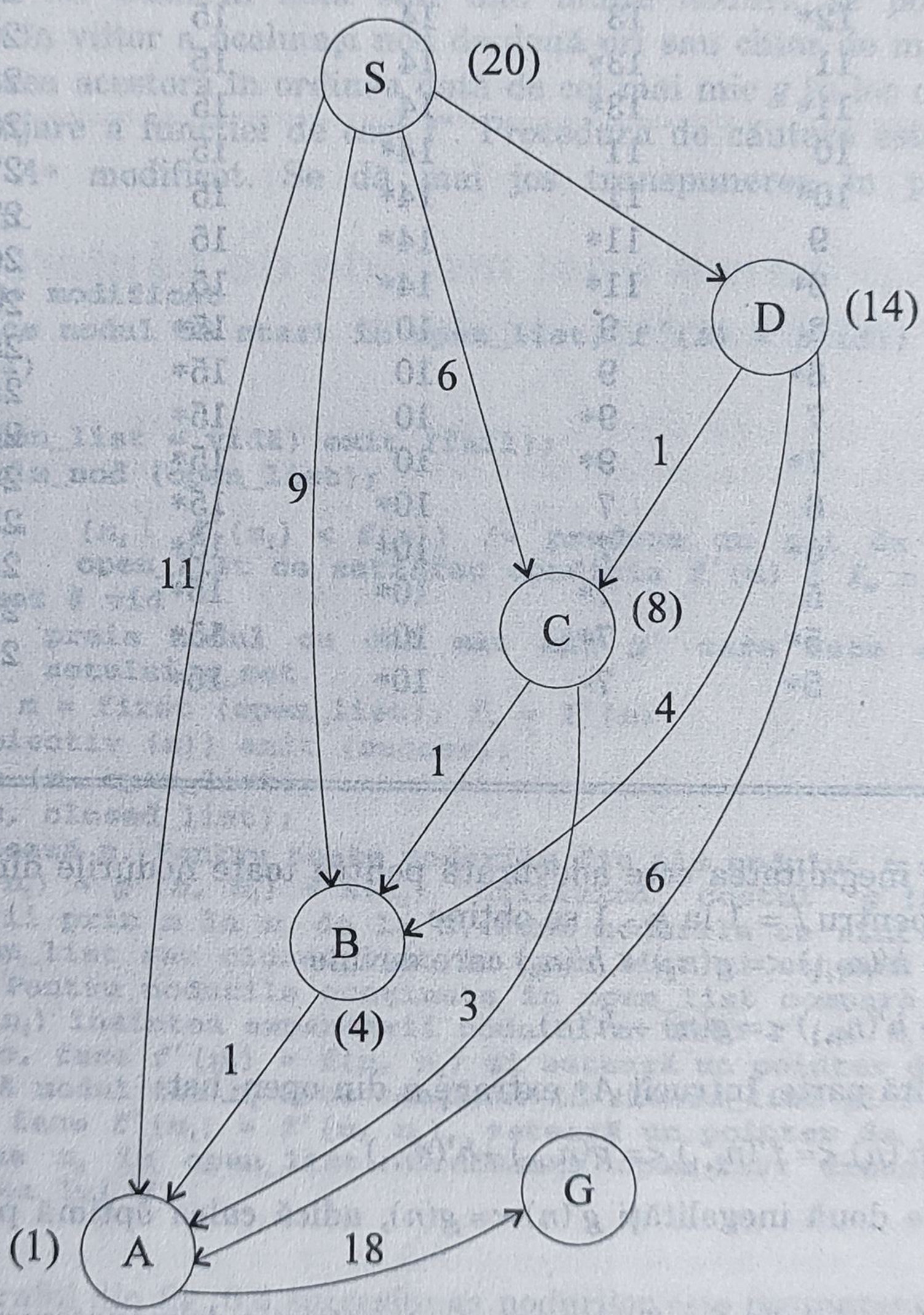
$$g(n_i) + h'(n_i) <= g(n_i) + h'(n_{i+1}) + c(n_i, n_{i+1})$$

Întrucît n_i şi n_{i+1} sînt conţinute în calea optimă

$g(n_{i+1}) = g(n_i) + c(n_i, n_{i+1})$ şi

$g(n_i) + h'(n_i) <= g(n_{i+1}) + h'(n_{i+1})$.

▼ ***Fig. 6.8. Graf în care un nod este expandat de mai multe ori prin algoritmul A****



▼ **Tabela 6.1**

S	A	B	C	D	G
20					
20*	12	13	14	15	29
20*	12*	13	14	15	29
20*	11	13*	14	15	28
20*	11*	13*	14	15	28
20*	10	11	14*	15	27
20*	10*	11	14*	15	27
20*	9	11*	14*	15	26
20*	9*	11*	14*	15	26
20*	8	9	10	15*	25
20*	8*	9	10	15*	25
20*	7	9*	10	15*	24
20*	7*	9*	10	15*	24
20*	6	7	10*	15*	23
20*	6*	7	10*	15*	23
20*	5	7*	10*	15*	22
20*	5*	7*	10*	15*	22*
20*	5*	7*	10*	15*	

Întrucît inegalitatea este adevărată pentru toate nodurile din P, dacă se utilizează pentru $l = 1$ la $k - 1$ se obţine

$g(n_{l+1}) + h'(n_{l+1}) <= g(n_k) + h'(n_k)$ care devine

$g(n_{l+1}) + h'(n_{l+1}) <= g(n) + h'(n)$.

Pe de altă parte, întrucît A^* extrage n din open_list

$g'(n) + h'(n) <= f'(n_{l+1}) <= g(n_{l+1}) + h'(n_{l+1})$

Din cele două inegalităţi $g'(n) <= g(n)$, adică calea optimă prin n a fost găsită.

Întrucît graful din fig. 6.8 nu satisface restricţia de monotonicitate, de mai multe ori un nod aflat pe drumul optim este expandat. Teorema 6.4 este aplicabilă dacă $h' = 0$ întrucît în acest caz restricţia de monotocitate este satisfăcută. În plus rezultă că teorema 6.2 este doar un caz particular al teoremei 6.5.

O îmbunătăţire a algoritmului prin care nodurile aflate pe drumul optim se expandează de mai puţine ori va fi prezentată. Ea se bazează pe faptul că înainte de terminarea căutării, un nod n ce satisface $f'(n) <= f(S)$ este prezent

în open_list, conform cu lema 6.2. Presupunînd că algoritmul A^* a expandat nodul n', se obţine $f'(n') <= f'(n) <= f(S)$. Dacă din nodurile expandate pînă la acest moment nodul avînd costul inferenţiat maxim înaintea expandării este m, $f'(m) <= f(S)$. Dacă $f'(n_i) < f'(m)$ pentru un nod n_i în open_list la acel moment, nodul n_i va fi inevitabil expandat. Raţiunea ce dictează aceasta arată că dacă $f'(n_i) < f'(m)$, se obţine $f'(m) < f(S)$ şi nodul scop $G(f'(G) = f(S))$ nu poate fi extras din open_list atîta timp cît nodul n_i este încă aici. De aceea vor fi expandate toate nodurile ce satisfac restricţia $f'(n_i) < f'(m)$ şi numai după aceea nodul m. Cum în listă sînt mai multe noduri, se poate preveni expandarea în viitor a aceluiaşi nod de două ori sau chiar de mai multe ori prin aranjarea acestora în ordinea dată de cel mai mic g în loc de utilizarea pentru aranjare a funcţiei de cost f'. Procedura de căutare este numită şi algoritmul A^* modificat. Se dă mai jos transpunerea în pseudocod a algoritmului.

```
Algoritm A* modificat
   Introduce nodul de start în open_list; f'(S) = h'(S);
   While (1)
   {
    if (open_list = vidă) exit (fail);
    n = prim_nod (open_list);
    n_set =   {n_i |  f'(n_i) < f(m)}  /* produce un set de noduri în
              open_list ce satisfac condiţia f'(m) < f_m n_set */
    if n_set # vid
      then  preia nodul cu cel mai mic g' care este element al
            setului n_set
      else n = first (open_list), f_m = f'(n)
    if (obiectiv (n)) exit (succes);
    remove (n, open_list);
    add (n, closed_list);
    expandează n. Pentru toate nodurile fiu ale nodului n calculează
    f'(n, n_i) = g'(n, n_i) + h(n_i)  utilizînd costul g'(n, n_i) al
    trecerii prin n la n_i de la S. Pune nodurile ce sînt conţinute
    în open_list sau closed_list în open_list şi setează pointerii
    la n. Pentru nodurile conţinute în open_list compară f'(n_i) şi
    f'(n, n_i) înaintea expandării nodului n. Dacă f'(n, n_i) este cel
    mai mic, face f'(n_i) = f(n, n_i) şi setează un pointer de la n_i la
    n. Dacă nodul fiu n_i este conţinut în closed_list şi f'(n, n_i) <
    f'(n_i) face f'(n_i) = f'(n, n_i), setează un pointer de la n la n_i
    şi pune n_i în open_list. Ordonează open_list crescător după
    valoarea lui f'.
   }
```

Pentru graful din fig. 6.8 succesiunea nodurilor este prezentată în tabelul 6.2. În tabel sînt prezentate nodurile din open_list şi closed_list cînd căutarea se face cu algoritmul A^* modificat. De asemenea sînt prezentate şi valorile funcţiilor de cost g', h', f_m. Pentru marcarea nodurilor din closed_list s-a utilizat simbolul *.

Se observă că se obţine o reducere considerabilă a numărului de expandări faţă de aplicarea algoritmului A^*. Se poate demonstra că în acest caz numărul maxim de expandări este n^2.

▼ **Tabelul 6.2**

S	A	B	C	D	G	f_m
0 + 20						0
0 + 20*	11 + 1	9 + 4	6 + 8	1 + 14		20
0 + 20	7 + 1	5 + 4	2 + 8	1 + 14*		20
0 + 20	5 + 1	3 + 4	2 + 8*	1 + 14*		20
0 + 20	4 + 1	3 + 4*	2 + 8*	1 + 14*	22 + 0	20
0 + 20	4 + 1*	3 + 4*	2 + 8*	1 + 14*	22 + 0*	20
0 + 20	4 + 1*	3 + 4*	2 + 8*	1 + 14*		22

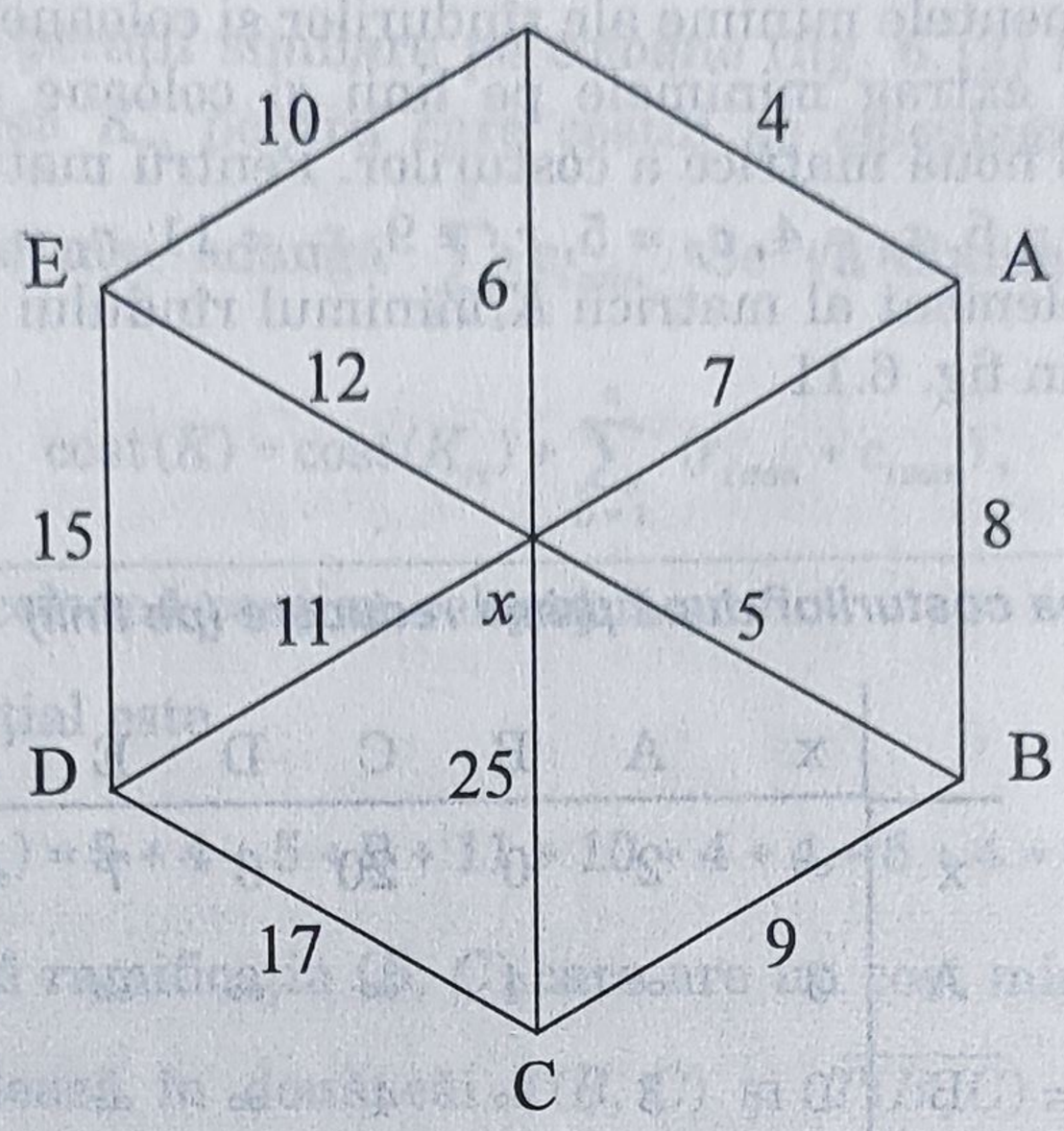
6.5. Aplicarea unei metode de căutare prin ramificare şi limitare

Metodele prezentate pînă în prezent nu se bazează pe informaţia de selecţie a alternativelor, decizia fiind luată pe baza corespondenţei nodului cu obiectivele problemei, motiv pentru care se mai numesc şi metode de cunoaştere informală. Îmbogăţirea informaţiei poate fi făcută prin ataşarea la fiecare nod a unei informaţii referitoare la costul trecerii de la o stare (nod părinte) la alta (nod succesor sau fiu).

O problemă veche ce se încearcă a fi rezolvată prin metode de căutare cu ramificare şi limitare este problema comis voiajorului. Formularea problemei arată că o persoană ce îndeplineşte, funcţia de comis voiajor şi lucrează într-un oraş X trebuie să ajungă odată pe lună într-o serie de oraşe în care are de dus obiecte sau are de luat anumite obiecte. Firma la care lucrează suportă costul transportului de la oraşul X la fiecare din oraşele de destinaţie. Problema comis voiajorului este de a stabili acel traseu pentru care pornind din oraşul X să atingă toate oraţele din traseu cu destinaţia finală oraşul X, astfel încît costul transportului să fie minim. În afară de informaţiile alternative ce reprezintă costul transportului între oraşul X şi oraşele învecinate sînt necesare informaţii despre costul deplasării între oraşele din traseu.

O primă componentă a metodei este cea referitoare la ramificare ce stabileşte metodele de obţinere a arborelui de căutare. Se poate considera ca informaţie de stare acea structură simbolică ce se pune în corespondenţă cu obiectivul problemei, structură ce ar fi dată de un circuit al oraşelor. Informaţia elementară poate fi privită ca informaţie de stare, fiind dată de conceptul de localitate, legăturile între localităţi fiind corespondentul direct al drumului fizic dintre acestea. Se va nota cu $T(i, j)$ submulţimea circuitelor ce conţine drumul elementar (i, j) ce se presupune a fi în drumul optim şi cu $\overline{T(i,j)}$ submulţimea circuitelor ce nu conţin drumul elementar (i, j). Astfel, dintre toate drumurile ce pornesc din nodul de start este probabil ca în soluţia optimă să se găsească acela care are costul cel mai mic dintre toate drumurile elementare ale problemei.

▼ *Fig. 6.9. Harta asociată oraşelor*



A doua componentă, cea de limitare se bazează pe faptul că fiecărui nod din spaţiu îi este ataşată o limită minimă a costului circuitului în mulţimea selectată de nodul respectiv. Dacă harta avînd costurile asociate este cea din fig. 6.9 se poate construi matricea costurilor (fig. 6.10). În strategia de reprezentare a matricii costurilor se pune pentru drumul între oraşul *A* şi *A* ca fiind ∞ întrucît odată ajuns aici acesta nu este un drum valid. La fel pentru oraşele între care nu există legătură directă se asociază costului infinit.

▼ *Fig. 6.10. Matricea costurilor*

K=

	x	A	B	C	D	E	F
x	∞	7	5	25	11	12	6
A	7	∞	8	∞	∞	∞	∞
B	5	8	∞	9	∞	∞	∞
C	25	∞	9	∞	17	∞	∞
D	11	∞	∞	17	∞	15	∞
E	12	∞	∞	∞	15	∞	10
F	6	4	∞	∞	∞	10	∞

Circuitul elementar are selectat pe fiecare rînd şi coloană cîte un drum elementar, soluţia problemei fiind acea secvenţă de drumuri ce asigură trecerea prin toate oraşele asigurînd în acelaşi timp cost minim. Limita minimă a costurilor pentru nodul iniţial se obţine considerînd că acesta conţine numai elementele minime ale rîndurilor şi coloanelor matricii de cost. Pentru aceasta se extrag minimele pe linii şi coloane din fiecare rînd şi coloană, obţinînd o nouă matrice a costurilor. Pentru matricea de mai sus se obţin pe rînduri $r_x = 5$, $r_A = 4$, $r_B = 5$, $r_C = 9$, $r_D = 11$, $r_E = 10$, $r_F = 4$. Dacă se scade din fiecare element al matricii K minimul rîndului respectiv se obţine matricea redusă din fig. 6.11.

▼ ***Fig. 6.11. Matricea costurilor după prima reducere (pe linii)***

		x	A	B	C	D	E	F
	x	∞	2	0	20	6	7	1
	A	3	∞	4	∞	∞	∞	0
Kr =	B	0	3	∞	4	∞	∞	∞
	C	16	∞	0	∞	8	∞	∞
	D	0	∞	∞	6	∞	4	∞
	E	7	∞	∞	∞	5	∞	0
	F	2	0	∞	∞	∞	6	∞

▼ ***Fig. 6.12. Matricea costurilor după a doua reducere (pe coloane)***

		x	A	B	C	D	E	F
	x	∞	2	0	16	1	3	1
	A	3	∞	4	∞	∞	∞	0
Krc =	B	0	3	∞	0	∞	∞	∞
	C	16	∞	0	∞	3	∞	∞
	D	0	∞	∞	2	∞	0	∞
	E	7	∞	∞	∞	0	∞	0
	F	2	0	∞	∞	∞	2	∞

Costul calculat pentru matricea K este echivalent cu cel calculat pentru matricea K_r dacă la acesta se adaugă $\sum_{L=1}^{n} r_{i\min}$.

Aplicarea unei operaţii similare pe coloane (fig. 6.12) face să se obţină o nouă matrice redusă K_{rc} pentru care costul se calculează la fel cu cel al matricii K_r însă se mai adaugă $\sum_{L=1}^{n} c_{i\min}$. Se va obţine limita minimă a costului ca fiind:

$$\text{cost}(K) = \text{cost}(K_{rc}) + \sum_{L=1}^{n} (r_{i\min} + c_{i\min}),$$

iar fiecare linie şi coloană conţine cel puţin un 0.

Astfel costul iniţial este

$C_0 = \sum_{L=1}^{n} (r_{i\min} + c_{i\min}) = 5 + 4 + 5 + 9 + 11 + 10 + 4 + 4 + 5 + 4 = 61$. Dacă din nodul iniţial se selectează ramificaţia (B, C) care are un cost minim în matricea K drumul se expandează în două căi $T(B, C)$ şi $\overline{T(B, C)}$. Calculul costului presupune eliminarea din matrice a rîndului B şi a coloanei C, indicînd faptul că odată oraşul atins nu se revine în el. De asemenea, se dă arcului (C, B) cost infinit în matrice, eliminînd astfel drumul de întoarcere. După eliminare se obţine matricea din fig. 6.13 asupra căreia se aplică o nouă operaţie de reducere dacă după eliminarea liniei şi coloanei au apărut linii sau coloane ce au minimul diferit de 0 (fig. 6.14).

▼ *Fig. 6.13. Matricea după eliminarea drumului (B, C)*

	x	A	B	D	E	F
x	∞	2	0	1	7	1
A	3	∞	4	∞	∞	0
C	16	∞	0	3	∞	∞
D	0	∞	∞	∞	0	∞
E	2	∞	∞	0	∞	0
F	2	0	∞	∞	2	∞

▼ ***Fig. 6.14. Noua matrice redusă***

	x	A	B	D	E	F
x	∞	2	0	1	7	1
A	3	∞	4	∞	∞	0
C	13	∞	∞	0	∞	∞
D	0	∞	∞	∞	0	∞
E	2	∞	∞	0	∞	0
F	2	0	∞	∞	2	∞

Prin reducerea matricii din fig. 6.13 al cărei rezultat este 6.14 se adaugă la costul minim valoarea 3 care este minimul pe linia C diferit de zero. Se obţine astfel $\text{cost}(T(B, C)) = 61 + 3 = 64$. Din acest moment se poate alege drumul elementar de cost minim (X, B), şi se elimină linia marcată cu X şi coloana marcată cu B. La acest moment nu este necesară reducerea, întrucît matricea costurilor obţinută conţine pe fiecare linie şi coloană cel puţin un zero. Se poate alege la acest moment traseul (F, A) de cost minim. La acest moment matricea costurilor se reduce după $r_A = 3$ obţinînd matricea din fig. 6.15. Costul asociat pînă în acest moment este $\text{cost}(T(B, C)) + 3 = 64 + 3 = 67$. La acest moment se poate alege drumul elementar (C, D), în care după alegerea drumului (D, E) de cost minim, rămîne o matrice de ordin 2 în care există două drumuri minime (A, X) şi (E, F).

▼ ***Fig. 6.15. Matrice redusă după mai mulţi paşi***

	x	D	E	F
A	0	∞	∞	∞
C	13	0	∞	∞
D	0	∞	0	∞
E	2	0	∞	0

Traseul de cost minim astfel obţinut este X, B, C, D, E, F, A, X avînd valoarea costului egală cu 67. La acest moment poate fi construit algoritmul general al căutării prin ramificare şi limitare corespunzător metodei de rezolvare a problemei comis voiajorului.

Pas1: se construieşte matricea costurilor corespunzătoare datelor iniţiale, stabilind pentru nodul iniţial mulţimea soluţiilor posibile şi limita costului minim cost(T);

Pas2: se obţine din matricea costurilor matricea redusă ce asignează costurile curente;

Pas3: se selectează un arc (i, j) pentru următoarea ramificaţie în arbore avînd drept criteriu de selecţie costul minim;

Pas4: se partitionează mulţimea soluţiilor posibile după drumurile $T(i, j)$ şi, $\overline{T(i,j)}$;

Pas5: se reduce matricea costurilor curente prin eliminarea rîndului i şi a coloanei j ce corespund arcului selectat (i, j), se calculează limitele maxime şi minime ale costului folosind relaţiile:

$$\text{cost}\,(T(i, j)) = \text{cost}\,(T \text{ anterior}) + \text{cost}\,(T(i, j))$$

$$\text{cost}\,\overline{T(i,j)} = \text{cost}\,(T \text{ anterior}) + \text{cost}\,\overline{T(i,j)}$$

în care, cost (T anterior) este limita minimă a costului nodului părinte din care s-a ramificat arcul (i, j), iar cost$(T(i, j))$ şi cost $\overline{T(i,j)}$ sînt costurile suplimentare ale circuitului optim în matricea redusă la pasul anterior.

Pas6: dacă ultimul arc a fost selectat, adică matricea costurilor are dimensiunea 2×2, s-a obţinut drumul optim;
altfel se reia procedura de la pasul 3.

Sînt situaţii în care matricea costurilor are termeni exprimaţi sub formă de variabile de stare, situaţie în care costul aplicării unui operator este determinat doar dacă se cunoaşte contextul de stare în care se aplică. În acest caz limita minimă a costului nu mai este reprezentată de suma celei mai reduse selecţii necorelate cu restricţiile problemei, ci se introduc corecţiile în selecţia operatorilor de ramificare. În momentul în care costul traseului pe ramificaţia selectată iniţial ca limită minimă a ajuns să depăşească valori neoptimale pînă atunci se selectează o altă ramificaţie, obţinînd condiţii suficiente de parcurs o altă ramificaţie mai avantajoasă faţă de cea anterioară.

7

REZOLVAREA PROBLEMELOR PRIN DECOMPOZIŢIE

O strategie de rezolvare a problemelor porneşte de la descompunerea acestora în probleme mai simple numite şi subprobleme. Dacă problema poate fi descompusă în subprobleme, spaţiul stărilor poate fi reprezentat prin grafuri ŞI/SAU, şi pot fi aplicate metodele de căutare specifice pentru grafuri ŞI/SAU metode ce includ căutarea prin minimizarea costului, precum şi metode de căutare euristică. Jocurile sînt probleme tipice reprezentate prin grafuri ŞI/SAU aşa că o strategie de mutare într-un joc este de fapt redusă la o problemă de căutare în graf.

7.1. Descompunerea problemelor

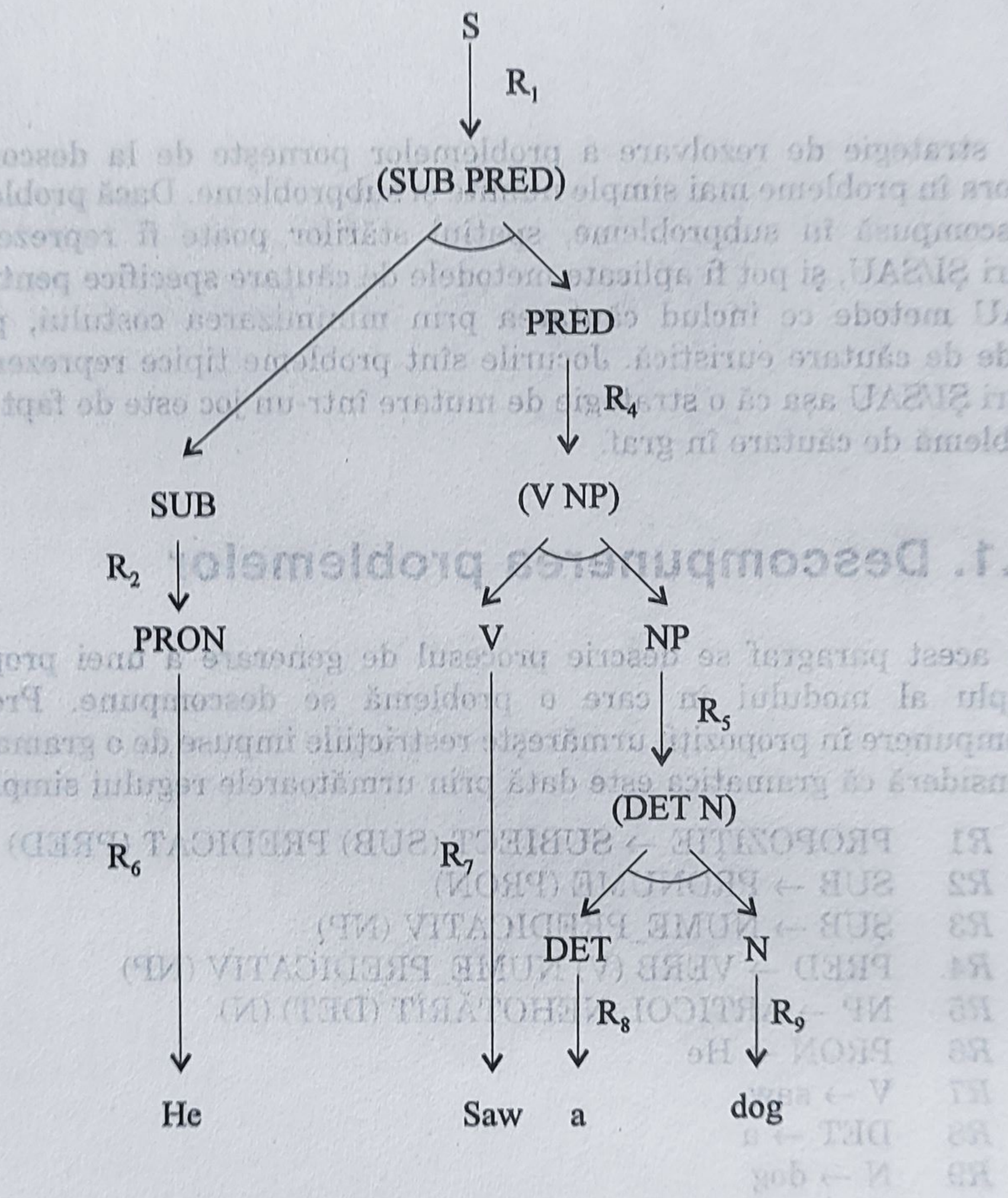
În acest paragraf se descrie procesul de generare a unei propoziţii, ca exemplu al modului în care o problemă se descompune. Procesul de descompunere în propoziţii urmăreşte restricţiile impuse de o gramatică dată. Se consideră că gramatica este dată prin următoarele regului simple:

$R1$	PROPOZIŢIE → SUBIECT (SUB) PREDICAT (PRED)
$R2$	SUB → PRONUME (PRON)
$R3$	SUB → NUME_PREDICATIV (NP)
$R4$	PRED → VERB (V) NUME_PREDICATIV (NP)
$R5$	NP → ARTICOL NEHOTĂRÎT (DET) (N)
$R6$	PRON → He
$R7$	V → saw
$R8$	DET → a
$R9$	N → dog
$R10$	N → cat

În stînga scrise cu litere mari s-au reprezentat simboluri neterminale, în dreapta în regulile $R6$-$R10$ sînt reprezentate cu litere mici simboluri terminale. Pornind de la starea iniţială S, se doreşte construirea unei propoziţii numai din simboluri terminale, prin aplicarea repetată a regulilor. O secvenţă de simboluri se poate specifica printr-o listă între paranteze.

Astfel prin aplicarea regulii $R1$, aplicabila în starea iniţială S se obţine (SUB PRED). Problema în acest moment este cea de conversie a listei într-o secvenţă de simboluri terminale. Subproblemele obţinute prin descompunerea problemei, adică cele de conversie într-o listă de simboluri terminale a elementelor SUB şi PRED, sînt mai simple şi pot fi rezolvare independent şi aplicate în paralel. Prin aplicarea regulii $R4$ la PRED se obţine lista (V NP), deci o descompunere în subproblemele V şi NP. Figura 7.1 arată un exemplu al procedurii pentru obţinerea unei soluţii.

▼ ***Fig. 7.1. Exemplu de generare a unei propoziţii***



Pentru a indica faptul că problema a fost descompusă în figură s-au specificat legături între drumurile grafului. Dacă se notează timpul pentru aplicarea unei reguli $t(R)$ şi diferite reguli pot fi aplicate independent, se poate presupune că sînt aplicate în paralel. Notînd cu $f(n)$ costul atingerii simbolului terminal de la nodul n se obţine:

$$f(\text{PRON}) = t(R6),\ f(V) = t(R7),\ f(\text{DET}) = t(R8),\ f(\text{N}) = t(R9)$$

Dacă se consideră că n este rezolvat prin nodurile fiu n_1, n_2,..., n_m se obţine:

$$f(n) = \max\{f(n_1), f(n_2),..., f(n_m)\}$$

Ca urmare, $f(\text{NP}) = t(R5) + \max\{f(\text{DET}), f(\text{N})\}$. Făcînd aceste calcule pentru toate nodurile se obţine costul total $f(S)$. Dacă regulile nu sînt aplicate în paralel se obţine pentru costul nodului n

$$f(n) = f(n_1) + f(n_2) + ... + f(n_m) = \sum_{i=1}^{m} f(n_i) .$$

Ultimele două relaţii sînt exemple tipice pentru calculul costului atunci cînd o problemă este descompusă. Una sau alta din aceste formule este practic utilizată la orice problemă. În calcul, nu s-a luat în considerare costul descompunerii, în cazul de faţă fiind considerat inclus în costul arcului dinaintea nodului corespunzînd descompunerii. Ca exemplu costul descompunerii (SUB PRED) este inclus în costul aplicării regulii $R1$.

7.2. Grafuri ŞI/SAU

Problema generării propoziţiei poate fi reprezentată prin grafuri ŞI/SAU ca în fig. 7.2. În acest caz graful este un arbore. La problemele reprezentate în spaţiul stărilor este suficient ca un singur nod soluţie să fie atins, pe cînd la grafurile ŞI/SAU procesul descompunerii desface problema în subprobleme.

Astfel în fig. 7.2 nodurile SUB şi PRED formează o structură de nod ŞI. Pentru rezolvarea problemei corespunzătoare nodului părinte al nodurilor ŞI, trebuiesc rezolvate toate subproblemele corespunzînd nodurilor fiu ale acestuia. În figură nodurile ŞI au fost marcate prin arce, nodurile nemarcate fiind noduri SAU. Nodurile SAU sînt identice cu cele prezentate în capitolul 6 referitoare la tehnici de căutare în arbori. Soluţia problemei cu structura din fig. 7.1 este o soluţie a grafului ŞI/SAU. Un graf parţial este numit *graf rezolvent,* iar rezolvarea unei probleme reprezentată prin grafuri ŞI/SAU invocă găsirea tuturor grafurilor rezolvente ale acesteia. În capitolul 6 au fost enumerate principalele caracteristici ale grafurilor ŞI/SAU. Graful din fig. 7.2 este un arbore ŞI/SAU. Dacă două noduri (cum este cazul nodului NP) sînt reprezentate ca un singur nod se obţine un graf ŞI/SAU, nu un arbore ŞI/SAU. Aici, intenţia de a reprezenta un nod de două ori este justificată de dorinţa de a obţine mai multe soluţii. În cazul de faţă soluţia "a dog saw a cat" a fost obţinută. În fig. 7.3a este prezentată o problemă complexă A ce poate fi rezolvată prin descompunerea în subproblemele B şi C sau prin rezolvarea lui D. În graful respectiv se regăsesc ambele tipuri de noduri pentru acelaşi părinte. În fig. 7.3b nodurile au fost separate prin introducerea unui nod suplimentar pentru care funcţia de cost este egală cu cea de descompunere a problemei A în subproblema M formată din B şi C.

Dacă se consideră că n este rezolvat prin nodurile fiu n_1, n_2,..., n_m se obţine:

$$f(n) = \max\{f(n_1), f(n_2),..., f(n_m)\}$$

Ca urmare, f(NP) = $t(R5)$ + max{f(DET), f(N)}. Făcînd aceste calcule pentru toate nodurile se obţine costul total $f(S)$. Dacă regulile nu sînt aplicate în paralel se obţine pentru costul nodului n

$$f(n) = f(n_1) + f(n_2) + ... + f(n_m) = \sum_{i=1}^{m} f(n_i) .$$

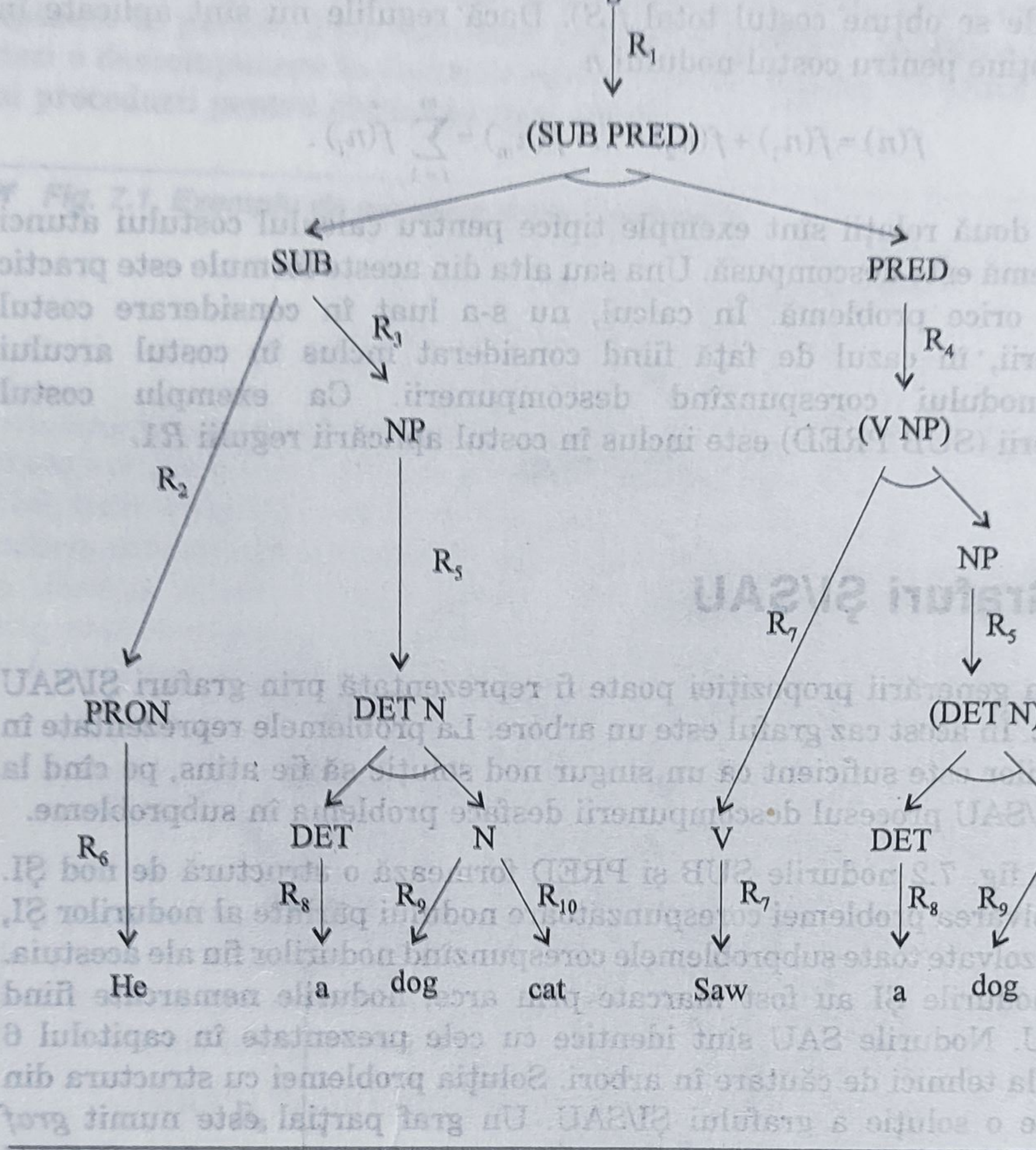
Ultimele două relaţii sînt exemple tipice pentru calculul costului atunci cînd o problemă este descompusă. Una sau alta din aceste formule este practic utilizată la orice problemă. În calcul, nu s-a luat în considerare costul descompunerii, în cazul de faţă fiind considerat inclus în costul arcului dinaintea nodului corespunzînd descompunerii. Ca exemplu costul descompunerii (SUB PRED) este inclus în costul aplicării regulii $R1$.

7.2. Grafuri ŞI/SAU

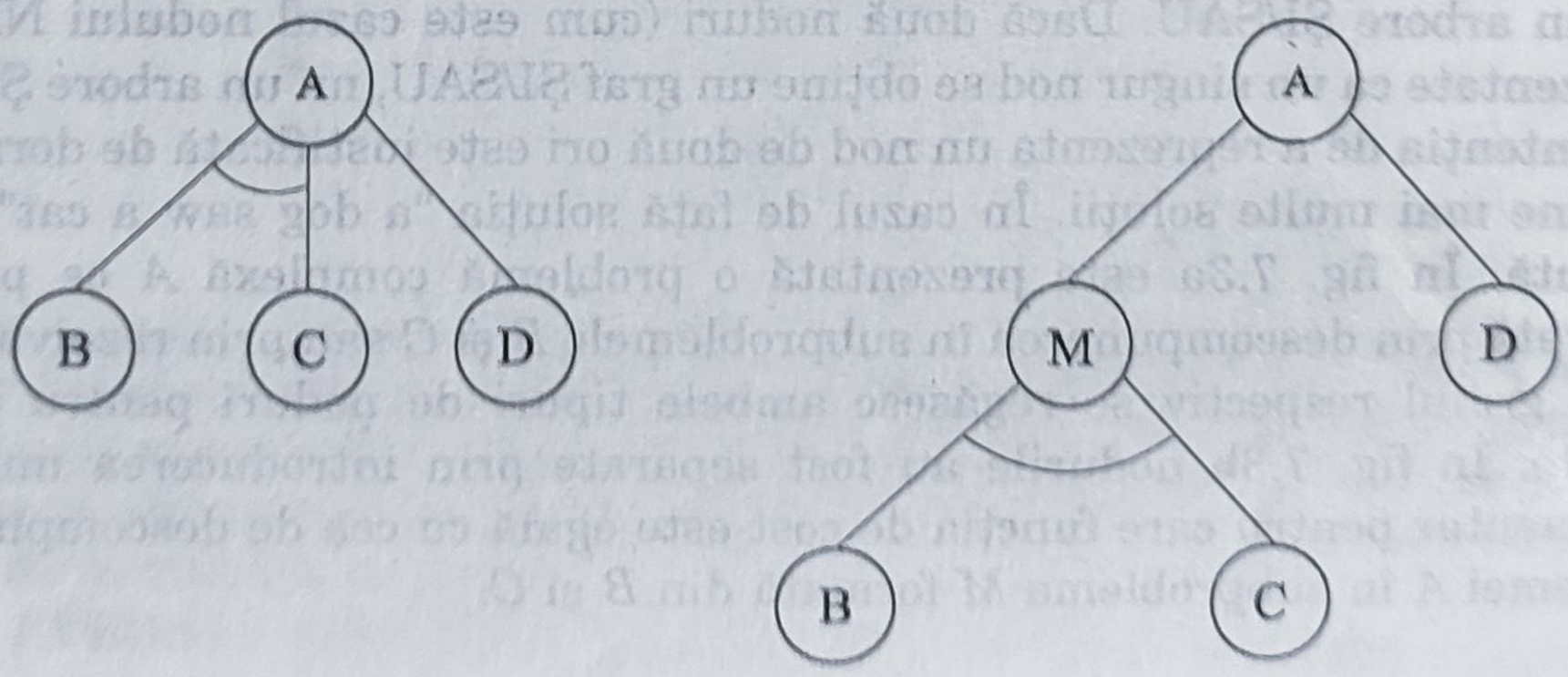
Problema generării propoziţiei poate fi reprezentată prin grafuri ŞI/SAU ca în fig. 7.2. În acest caz graful este un arbore. La problemele reprezentate în spaţiul stărilor este suficient ca un singur nod soluţie să fie atins, pe cînd la grafurile ŞI/SAU procesul descompunerii desface problema în subprobleme.

Astfel în fig. 7.2 nodurile SUB şi PRED formează o structură de nod ŞI. Pentru rezolvarea problemei corespunzătoare nodului părinte al nodurilor ŞI, trebuiesc rezolvate toate subproblemele corespunzînd nodurilor fiu ale acestuia. În figură nodurile ŞI au fost marcate prin arce, nodurile nemarcate fiind noduri SAU. Nodurile SAU sînt identice cu cele prezentate în capitolul 6 referitoare la tehnici de căutare în arbori. Soluţia problemei cu structura din fig. 7.1 este o soluţie a grafului ŞI/SAU. Un graf parţial este numit *graf rezolvent,* iar rezolvarea unei probleme reprezentată prin grafuri ŞI/SAU invocă găsirea tuturor grafurilor rezolvente ale acesteia. În capitolul 6 au fost enumerate principalele caracteristici ale grafurilor ŞI/SAU. Graful din fig. 7.2 este un arbore ŞI/SAU. Dacă două noduri (cum este cazul nodului NP) sînt reprezentate ca un singur nod se obţine un graf ŞI/SAU, nu un arbore ŞI/SAU. Aici, intenţia de a reprezenta un nod de două ori este justificată de dorinţa de a obţine mai multe soluţii. În cazul de faţă soluţia "a dog saw a cat" a fost obţinută. În fig. 7.3a este prezentată o problemă complexă A ce poate fi rezolvată prin descompunerea în subproblemele B şi C sau prin rezolvarea lui D. În graful respectiv se regăsesc ambele tipuri de noduri pentru acelaşi părinte. În fig. 7.3b nodurile au fost separate prin introducerea unui nod suplimentar pentru care funcţia de cost este egală cu cea de descompunere a problemei A în subproblema M formată din B şi C.

▼ *Fig. 7.2. Graf ŞI/SAU al problemei de generare a propoziţiei*



▼ *Fig. 7.3. Metode de reprezentare a nodurilor ŞI/SAU mixate*
a) noduri ŞI, SAU sînt fii ale aceluiaşi nod
b) graf ŞI/SAU



Un nod într-un arbore ŞI/SAU poate fi un nod ŞI al unui nod părinte dat şi în acelaşi timp un nod SAU al altui nod părinte. Ca urmare, este necesar să se identifice relaţia părinte fiu. În acest material se vor utiliza expresiile "n_i este un nod ŞI al nodului n" sau "nodul n_i al lui n este un nod ŞI".

7.3. Căutarea în grafuri ŞI/SAU

Se tratează în continuare deferite metode pentru găsirea soluţiei într-un graf ŞI/SAU. Cînd se caută într-un graf este suficient să se găsească un singur drum, aşa că dacă se cunosc pointerii de la nodurile părinte către toţi fii săi în drumul parcurs, şi căutarea este completă, înseamnă că s-a găsit o cale ce constituie o soluţie. Totuşi un graf rezolvent al grafului ŞI/SAU este la rîndul său un graf. Procesul găsirii soluţiei prin căutare corespunde dezvoltării progresive a soluţiei. Numai unele din grafurile rezolvente posibile sînt găsite în timpul căutării. Acestea sînt numite *grafuri candidate.* S-a preferat această denumire în loc de *grafuri rezolvente parţiale* întrucît nu se cunoaşte încă dacă acestea sînt parte a soluţiei. Uzual aceste grafuri sînt găsite în timpul căutării şi structurile lor trebuiesc reprezentate.

7.3.1. Evaluarea şi expandarea grafurilor candidate

Un graf ŞI/SAU poate fi cercetat prin evaluări şi expandări repetate după cum urmează:

1. Determină dacă nodurile grafurilor candidate sînt solvabile, insolvabile sau nedecidabile, precum şi costul acestora dacă este cazul.
2. Expandează grafurile candidate.

În primul pas examinează toate punctele finale (nodurile frunză) ale grafului candidat. Dacă acestea sînt noduri terminale ele corespund unei probleme rezolvate şi ca urmare vor fi marcate în acest sens. Dacă punctele terminale ale grafului candidat sînt puncte finale ale problemei iniţiale, dar nu noduri terminale, problemele corespunzătoare acestor noduri sînt insolvabile şi se vor marca printr-un marker UNSOLVED. Dacă nodurile fiu ale unui nod, altele decît nodurile terminale ale grafului candidat sînt noduri ŞI, şi dacă toţi fii sînt SOLVED, atunci nodul va fi marcat SOLVED. Dacă cel puţin un fiu este UNSOLVED, atunci nodul părinte va fi marcat ca UNSOLVED. În cazul în care nodurile fiu, altele decît punctele terminale sînt noduri SAU şi cel puţin unul a fost marcat solved atunci nodul părinte va purta aceeaşi marcă.

Cînd nodul S al unui graf candidat devine SOLVED, graful este o soluţie a problemei. Dacă este necesar calculul costului, se calculează costul tuturor nodurilor utilizînd costul punctelor finale ale grafurilor candidate. Costul nodului părinte al unui nod ŞI se calculează ca suma costurilor tuturor nodurilor sale fiu, pe cînd cel al unui nod SAU ca maximul costurilor fiilor. Costul nodului de start este costul grafului candidat. În pasul 2 se găsesc nodurile fiu prin expandarea nodurilor ce sînt puncte terminale ale grafului

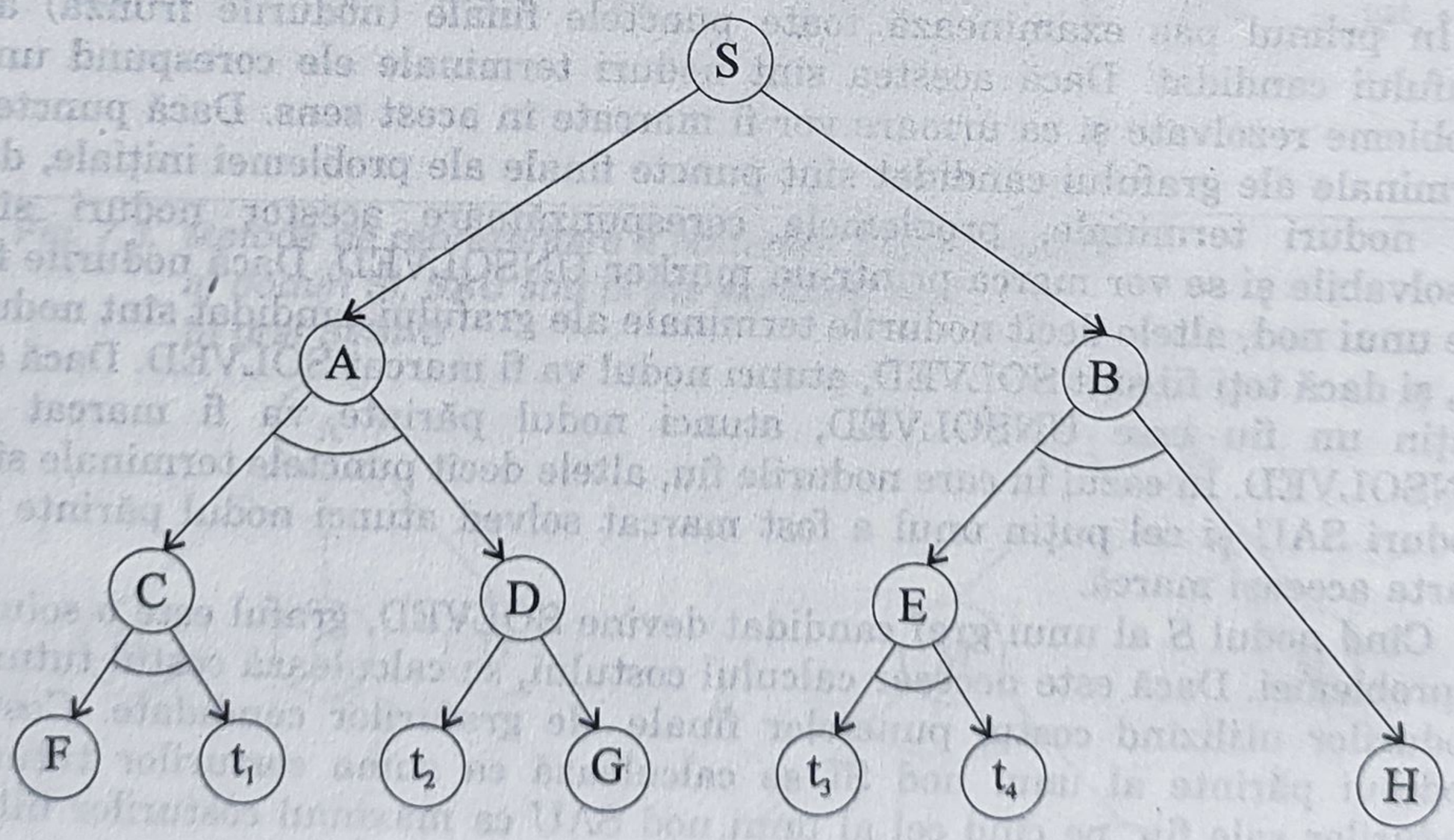
candidat p şi nu au fost încă marcate în nici un fel. Dacă nodurile fiu ale nodului expandat n sînt noduri ŞI un nou graf candidat este generat prin adăugarea tuturor nodurilor fiu la nodul p. Dacă nodurile fiu ale nodului n sînt noduri SAU adăugarea cîte unuia din acestea la p va determina generarea mai multor grafuri cîte noduri fiu are. În acest mod un mare număr de grafuri candidate vor fi generate prin procesul de căutare.

7.3.2. Căutarea în adîncime în grafuri ŞI/SAU

Ca şi în cazul căutării în grafuri obişnuite, strategiile pentru căutarea în grafurile ŞI/SAU sînt: căutare în adîncime, căutare în lărgime, strategii de căutare optimală. Procedura de căutare în adîncime este prezentată în pseudocod mai jos.

```
Introduce în open_list un graf candidat format doar din nodul de
start
  While (1)
  {
    if (open_list = vidă) exit (fail);
    p = prim_graf (open_list); /* aduce primul graf din listă
    if (solved (p)) exit (succes); /* dacă p este un graf rezolvent
       căutarea se termină şi p este soluţia */
    remove (p, open_list)
    add (n, closed_list);
    expandează p, evaluează toate noile grafuri candidate şi pune
    grafurile candidate în care nodul de start nu este UNSOLVED în
    open_list;
  }
```

▼ *Fig. 7.4. Căutarea în grafuri ŞI/SAU, t_1 - t_4 sînt noduri terminale*



▼ ***Fig. 7.5. Grafuri în open_list la căutarea în adîncime***

În fig. 7.5 se arată grafurile candidate în open_list pentru graful ŞI/SAU din figura 7.4. Cînd primul graf candidat în open_list este expandat şi evaluat la a treia parcurgere a buclei este găsit UNSOLVED şi este mutat din open_list. La a şasea repetare a procedurii graful candidat din open_list este SOLVED şi devine soluţie a problemei.

În această procedură grafurile candidate expandate sînt ţinute în open_list, însă nu este esenţial a face acest lucru în cazul căutării în adîncime. Se poate utiliza un sistem în care această listă conţine numai un graf parţial şi dacă starea sa este în final SOLVED, pentru formarea soluţiei trebuie combinat corespunzător cu alte grafuri. Pentru realizarea acestui lucru este necesară construirea unei funcţii care atunci cînd nodurile sînt expandate toate nodurile fiu ale sale au fost generate o singură dată. Un singur nod fiu determină un graf parţial la un moment de timp. Pentru formalizare se defineşte funcţia *next_fiu*. Să presupunem că nodul n are nodurile fiu $n_1, n_2, \ldots, n_m$ şi nodul fiu n_i a fost deja generat. Dacă nici un nod fiu nu a fost deja generat $i = 0$. Aplicarea funcţiei next_fiu(n) la nodul n determină următorul rezultat

$$\text{next\_fiu}(n) = \begin{cases} n_{i+1}, & 0 <= i <= m \\ \text{NULL}, & i = m \end{cases}.$$

Cu acest observaţii algoritmul de căutare în adîncime în arbori ŞI/SAU se desfăşoară după procedura:

```
   Construieşte graful candidat p conţinînd numai nodul de   start
n = s;
   Evaluează p;
while true
{
  while true
  {
    if unsolved (p) exit (fail);
    if solved (p) exit (succes);
    if n este marcat break;
    next_n = next_fiu (n)
    add next_n la p, evaluează p;
    n = next_n;
  }
  evaluează p
  if n este UNSOLVED elimină n din graful p;
  n = nod_părinte (n);
}
```

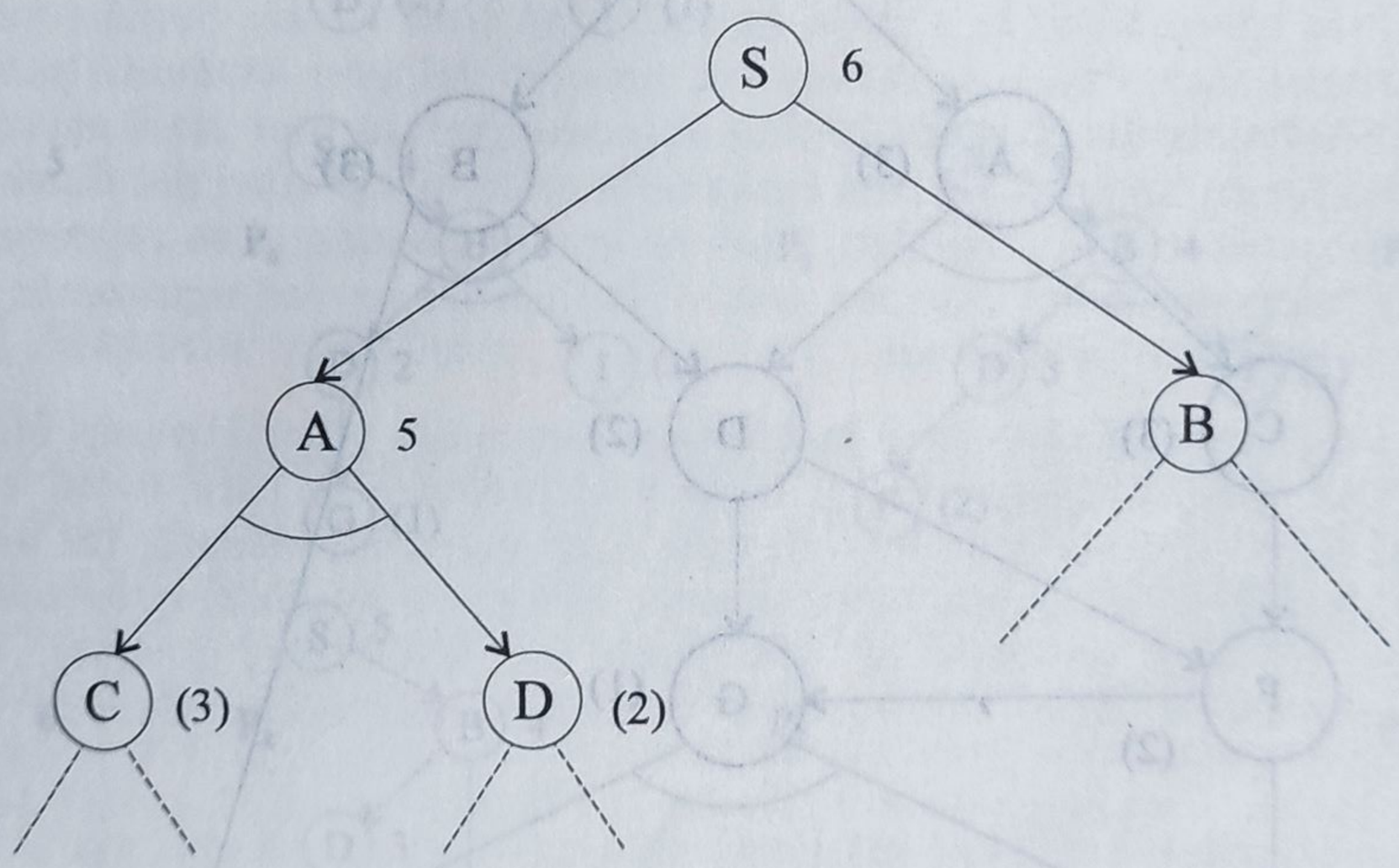
Aplicată la graful din fig. 7.4 se obţine secvenţa $S, A, C, F, C, A, S, B, E, t_3, E, t_4$. Secvenţa de căutare este aceeaşi cu cea de căutare în adîncime în grafuri obişnuite. Dacă în procesul de căutare se găseşte ca toate nodurile SAU sînt UNSOLVED se întoarce la nodul imediat precedent şi continuă căutarea după alte arce. Acest proces este numit *backtrack*. Întrucît procedura realizează funcţia backtrack nu este necesar ca toate nodurile fiu expandate ale unui nod să fie stocate în open_list. Dacă într-un graf candidat p nodurile fiu ale nodului final n sînt într-un graf candidat p, şi nodurile fiu ale nodului final n sînt noduri ŞI se adaugă toate la graful p. Pentru a reduce timpul cînd unul din nodurile fiu este UNSOLVED n devine UNSOLVED şi nu necesită găsirea altor noduri fiu. Pentru a continua procesul se utilizează funcţia next_fiu.

7.3.3. Căutarea soluţiei optime în grafuri ŞI/SAU

În acest paragraf se tratează găsirea soluţiei în graf, soluţie avînd cost minim cînd costul arcelor către nodurile SAU este definit. Se presupune în continuare că pentru fiecare nod al grafului costul estimat al soluţiei este dat. Aceasta corespunde la h' în căutarea pentru grafuri obişnuite. Se va nota costul estimat pentru nodul n cu $h'(n)$. Dacă funcţia euristică $h'(n)$ nu este dată, cazul este similar cu $h'(n) = 0$ şi algoritmul este identic cu cel de căutare în adîncime pentru grafuri ŞI/SAU.

La algoritmul de căutare în adîncime descris în precedenta secţiune diferite grafuri candidate au fost generate. Valoarea inferenţială $f'(p)$ a costului unui graf candidat p este obţinută prin evaluarea lui p utilizînd valoarea inferenţiată a costului punctelor finale. În fig. 7.6 se arată o parte a unui graf ŞI/SAU. Un graf candidat p este prezentat prin linii pline. Costul nodurilor ŞI se calculează utilizînd suma costurilor fiilor săi, iar pentru nodurile SAU s-a luat valoarea 1.

▼ ***Fig. 7.6. Estimarea costului în graf candidat. Între paranteze s-a prezentat h′ şi fără paranteze costurile evaluate***

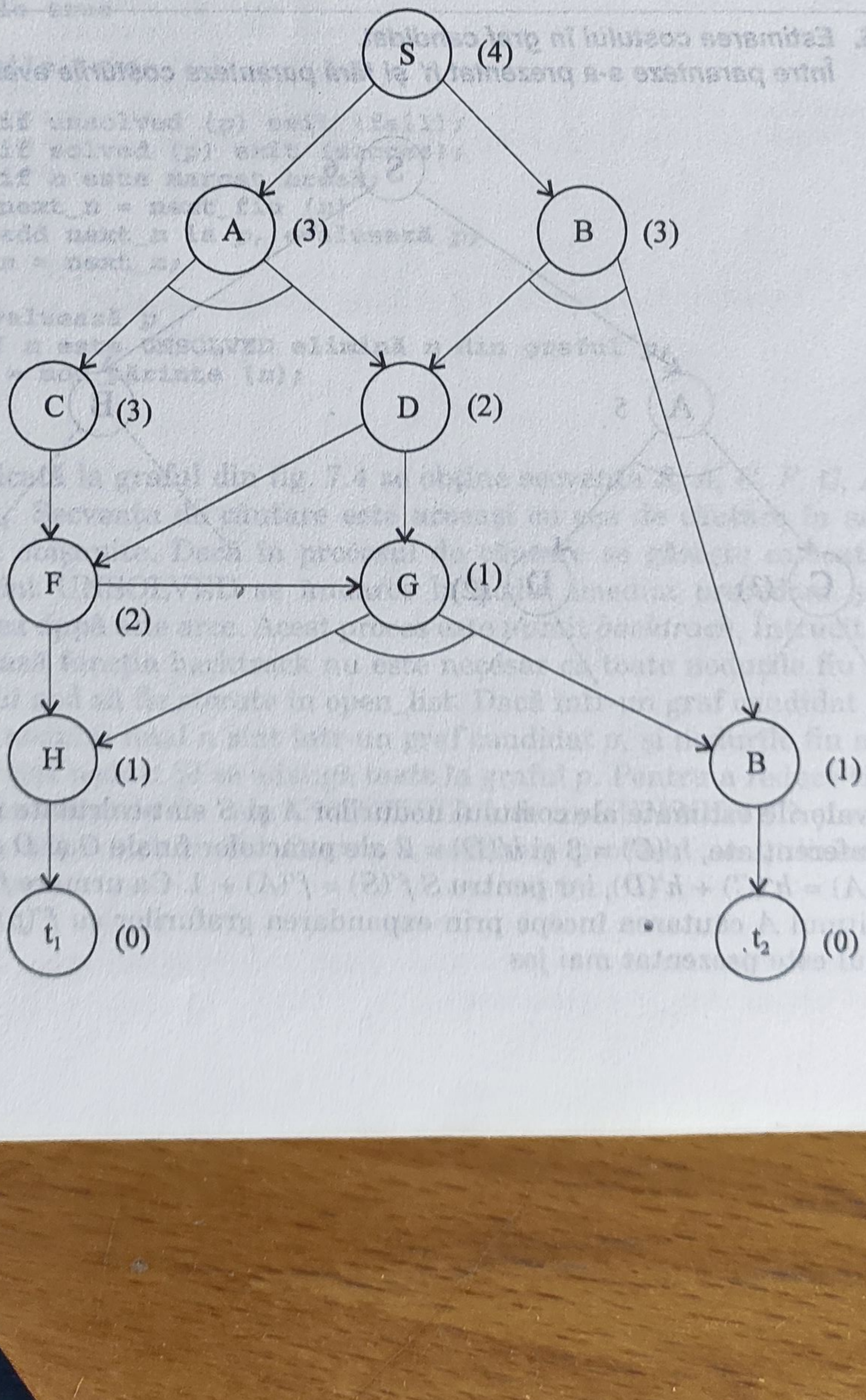


Dacă valorile estimate ale costului nodurilor A şi S sînt calculate utilizînd valorile inferenţiate, $h'(C) = 3$ şi $h'(D) = 2$ ale punctelor finale C şi D din p, se obţine $f'(A) = h'(C) + h'(D)$, iar pentru S $f'(S) = f'(A) + 1$. Ca urmare $f'(p) = 6$. Cu algoritmul A căutarea începe prin expandarea grafurilor cu $f'(p)$ minim. Algoritmul este prezentat mai jos

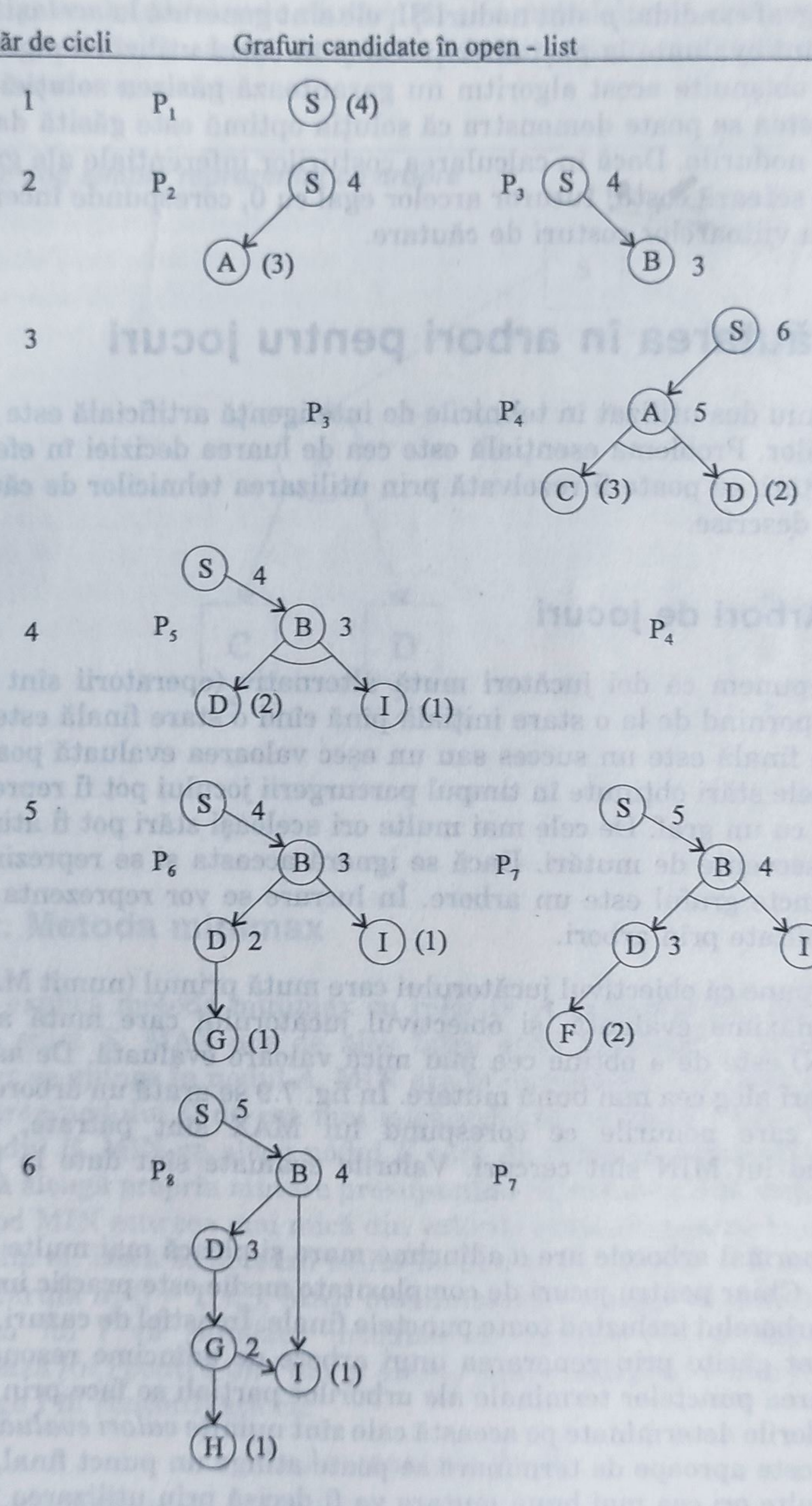
```
   Pune în open_list graful candidat p format numai din nodul    de
start S
                                   f'(p) = h'(S);
   while true
   {
     if (open_list = vidă) exit (fail);
     p = prim_graf (open_list); /* aduce primul graf din listă
     if (solved (p)) exit (succes); /* dacă p este un graf rezolvent
        căutarea se termină şi p este soluţia */
     remove (p, open_list).
     expandează p, evaluează toate grafurile candidate p_i nou generate
     şi găseşte f'(p_i). Dacă p_i este UNSOLVED îl pune în open_list.
   }
```

Pentru a ilustra modul în care algoritmul avansează se consideră exemplul din fig. 7.7, iar în fig. 7.8 se prezintă conţinutul în open_list prin utilizarea algoritmului de căutare optimală. În figuri s-au trecut între paranteze costurile estimate, costurile fără paranteze sînt calculate utilizînd costuri estimate. Fiecare graf candidat este denumit, pentru a nu desena de mai multe ori acelaşi graf.

▼ ***Fig. 7.7. Căutarea soluţiei optime în grafuri ŞI/SAU***



▼ ***Fig. 7.8. Soluția căutării optimale pentru graful din fig. 7.7***



În fig. 7.8 sînt arătaţi numai primii şase ciclii. După ultimul ciclu ilustrat, H şi I sînt expandate, şi nodurile terminale t_1 şi t_2 sînt adăugate obţinînd graful p_{10}. Costul pentru p_{10} este $f(p_{10}) = 5$. Dacă nodurile fiu ale unui nod n asociat unui graf candidat p sînt noduri ŞI, ele sînt generate la acelaşi moment cu toate că sînt evaluate la paşi diferiţi. Ca şi în cazul utilizării algoritmului A în grafuri obişnuite acest algoritm nu garantează găsirea soluţiei optime. Cu toate acestea se poate demonstra că soluţia optimă este găsită dacă $h' < h$ pentru toate nodurile. Dacă în calcularea costurilor inferentiale ale grafurilor candidate se setează costul tuturor arcelor egal cu 0, corespunde încercării de minimizare a viitoarelor costuri de căutare.

7.4. Căutarea în arbori pentru jocuri

Un domeniu des utilizat în tehnicile de inteligenţă artificială este şi cel al teoriei jocurilor. Problema esenţială este cea de luarea deciziei în efectuarea unei noi mutări, ce poate fi rezolvată prin utilizarea tehnicilor de căutare ce au fost deja descrise.

7.4.1. Arbori de jocuri

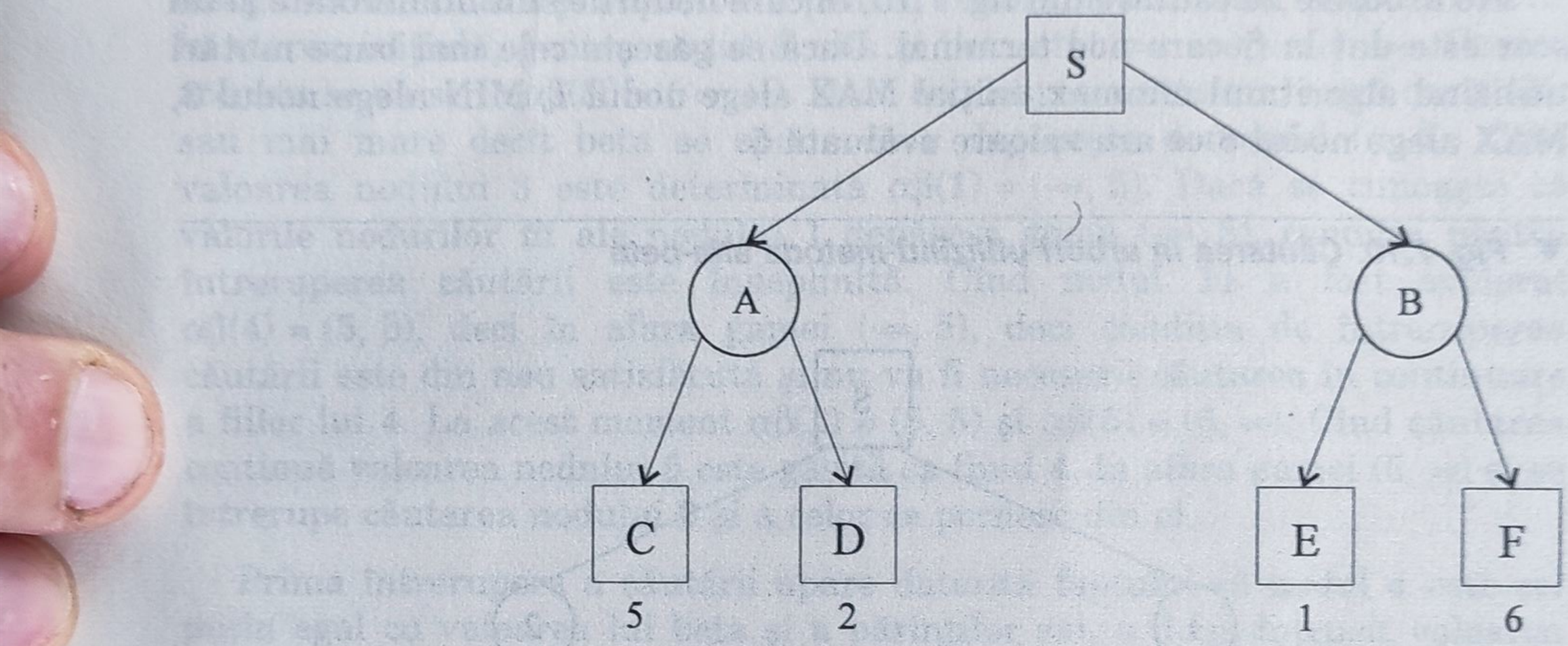
Să presupunem că doi jucători mută alternativ (operatorii sînt aplicaţi alternativ), pornind de la o stare iniţială pînă cînd o stare finală este atinsă. Dacă starea finală este un succes sau un eşec valoarea evaluată poate fi un scor. Diferitele stări obţinute în timpul parcurgerii jocului pot fi reprezentate asemănător cu un graf. De cele mai multe ori aceleaşi stări pot fi atinse prin mai multe secvenţe de mutări. Dacă se ignoră aceasta şi se reprezintă prin noduri distincte graful este un arbore. În lucrare se vor reprezenta jocurile pentru simplitate prin arbori.

Se presupune că obiectivul jucătorului care mută primul (numit MAX) este să obţină maxima evaluată, şi obiectivul jucătorului care mută al doilea (numit MIN) este de a obţine cea mai mică valoare evaluată. De asemenea ambii jucători aleg cea mai bună mutare. În fig. 7.9 se arată un arbore asociat simplu, în care nodurile ce corespund lui MAX sînt pătrate, iar cele corespunzînd lui MIN sînt cercuri. Valorile evaluate sînt date la punctele finale.

În mod normal arborele are o adîncime mare şi pleacă mai multe arce din fiecare nod. Chiar pentru jocuri de complexitate medie este practic imposibilă generarea arborelui incluzînd toate punctele finale. În astfel de cazuri, valorile evaluate sînt găsite prin generarea unui arbore de adîncime rezonabilă, în care evaluarea punctelor terminale ale arborilor parţiali se face prin aceleaşi metode. Valorile determinate pe această cale sînt numite *valori evaluate static*. Dacă jocul este aproape de terminare se poate atinge un punct final, însă de cele mai multe ori cea mai bună mutare va fi decisă prin utilizarea valorilor evaluate static. Se ia în consideraţie faptul că adversarul determină şi el cea mai bună mutare după generarea unui arbore parţial de adîncime

corespunzătoare. Adîncimea arborelui este restricţionată prin memoria şi timpul de calcul necesare. Valorile evaluate static sînt date de cunoaşterea asupra caracteristicilor jocului. Pentru a juca şah, Shogi sau go este necesară atît investigarea metodelor de căutare cît şi a metodelor de evaluare adecvate. În lucrare se prezintă numai metode de căutare atunci cînd valorile evaluate sînt considerate cunoscute.

▼ *Fig. 7.9. Joc simplu reprezentat ca arbore*



7.4.2. Metoda minimax

Se va explica metoda minimax cu referire la exemplul din fig. 7.9. De la nodul de start *S*, MAX are de ales între două posibilităţi. Dacă se alege primul arc se ajunge în nodul *A*. MIN are în continuare alte două posibilităţi, dar alegerea nodului *D* dă cea mai mică valoare evaluată. Dacă MAX a ales iniţial nodul *B*, MIN va alege nodul *E* care dă valoarea inferenţiată 1. MAX trebuie să aleagă propria mutare presupunînd că valoarea inferenţiată pentru fiecare nod MIN este cea mai mică din valorile evaluate pentru toate nodurile pornind din el. Dacă se notează nodul MAX n şi fii săi n_i ($i = 1, m$), iar nodul fiu al fiecăruia n_{ij} ($j = 1, i_m$), MAX maximizează valoarea evaluată a lui $f(n_i)$. Selectarea lui i va satisface următoarea condiţie $f(n_i) = \max f(n_i)$. MIN minimizează $f(n_i)$ pentru un i dat şi alege j astfel încît $f(n_i) = \min f(n_{ij})$. Aşa că MAX alege i în concordanţă cu:

$$f(n_{ij}) = \max\{\min f(n_{ij})\}.$$

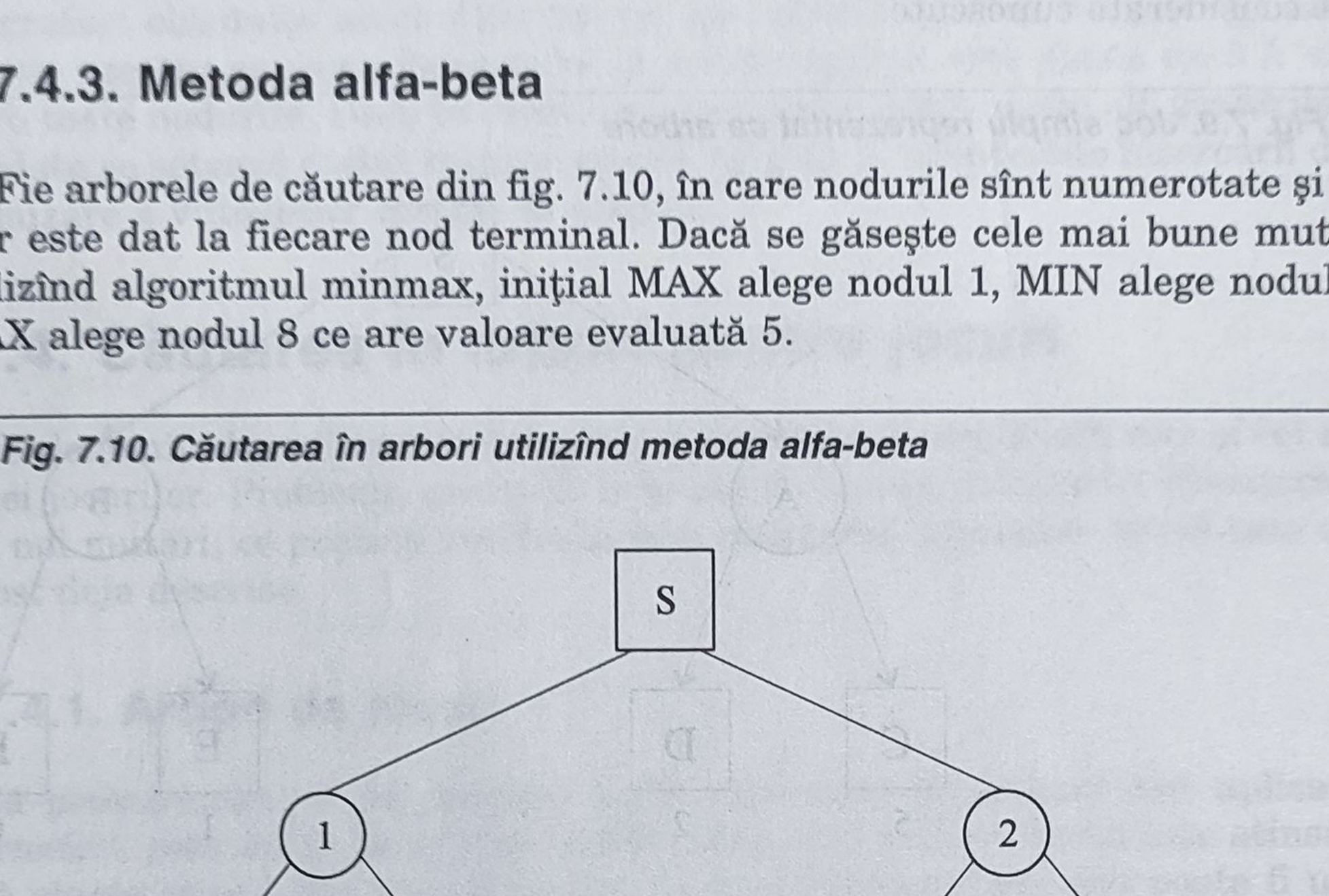
Dacă se ia adîncimea arborelui k, MAX va selecta i_1 aşa că

$$f(n_{i1, i2,..., ik}) = \max \min \ldots \{f(n_{i1, i2,..., ik})\}$$

Dacă valorile evaluate ale punctelor terminale din arbore sînt corecte, metoda minimax va selecta cea mai bună mutare. Totuşi, dacă adîncimea arborelui creşte, numărul de noduri ce trebuiesc evaluate creşte exponenţial. Cercetările actuale caută să găsească cel mai eficient algoritm. Un exemplu tipic este explicat în paragraful 7.4.3.

7.4.3. Metoda alfa-beta

Fie arborele de căutare din fig. 7.10, în care nodurile sînt numerotate şi un scor este dat la fiecare nod terminal. Dacă se găseşte cele mai bune mutări utilizînd algoritmul minmax, iniţial MAX alege nodul 1, MIN alege nodul 3, MAX alege nodul 8 ce are valoare evaluată 5.



▼ ***Fig. 7.10. Căutarea în arbori utilizînd metoda alfa-beta***

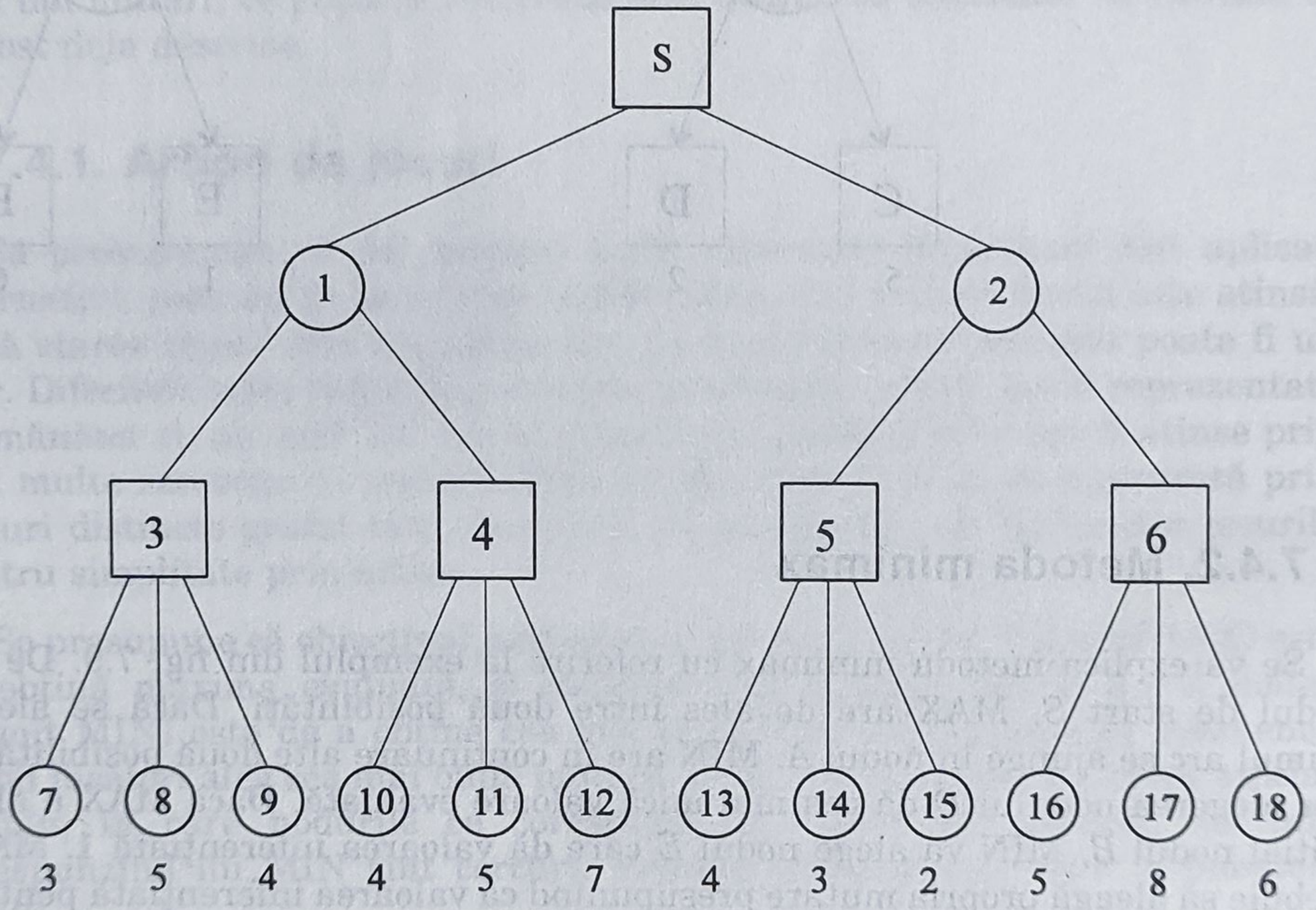
Presupunînd că într-o procedură ce este similară cu cea de căutare în adîncime, s-a obţinut valoarea evaluată 5 pentru nodul 3, în continuare se încearcă găsirea valorii nodului 4. Se cunoaşte la aceste moment faptul că valoarea nodului 1 nu poate fi mai mare decît 5. Motivul rezidă în faptul că dacă valoarea nodului 4 ar fi mai mare decît 5, MIN nu va alege nodul 4. Cu toate că se cunoaşte că, valorile nodurilor 10 şi 11 sînt obţinute, valoarea nodului 4 este cel puţin 5. Aşa că nu poate fi mai avantajos pentru MIN să aleagă nodul 4 în detrimentul nodului 3. Se deduce că MIN va alege nodul 3

Dacă valorile evaluate ale punctelor terminale din arbore sînt corecte, metoda minimax va selecta cea mai bună mutare. Totuşi, dacă adîncimea arborelui creşte, numărul de noduri ce trebuiesc evaluate creşte exponenţial. Cercetările actuale caută să găsească cel mai eficient algoritm. Un exemplu tipic este explicat în paragraful 7.4.3.

7.4.3. Metoda alfa-beta

Fie arborele de căutare din fig. 7.10, în care nodurile sînt numerotate şi un scor este dat la fiecare nod terminal. Dacă se găseşte cele mai bune mutări utilizînd algoritmul minmax, iniţial MAX alege nodul 1, MIN alege nodul 3, MAX alege nodul 8 ce are valoare evaluată 5.

▼ ***Fig. 7.10. Căutarea în arbori utilizînd metoda alfa-beta***



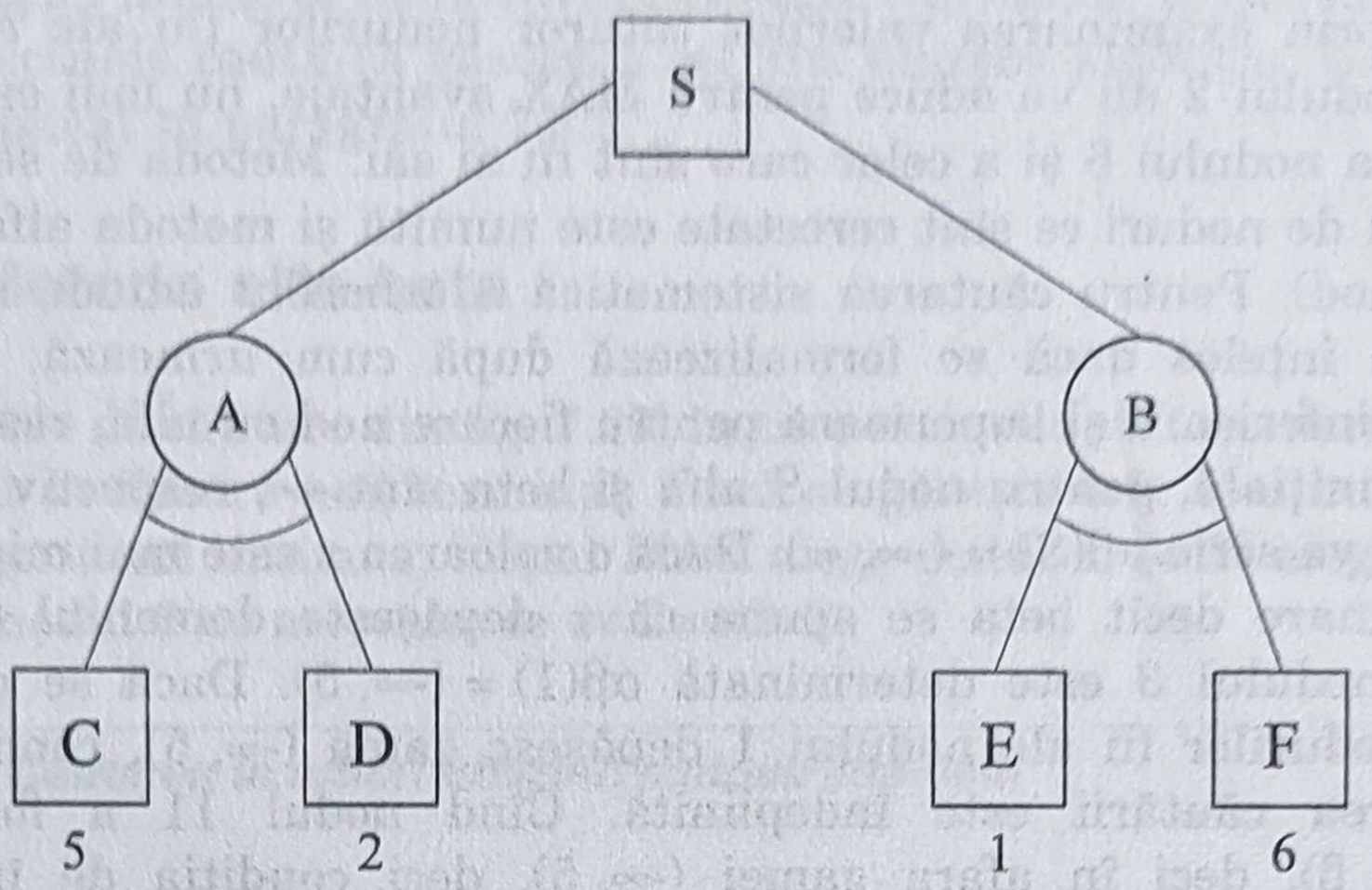
Presupunînd că într-o procedură ce este similară cu cea de căutare în adîncime, s-a obţinut valoarea evaluată 5 pentru nodul 3, în continuare se încearcă găsirea valorii nodului 4. Se cunoaşte la aceste moment faptul că valoarea nodului 1 nu poate fi mai mare decît 5. Motivul rezidă în faptul că dacă valoarea nodului 4 ar fi mai mare decît 5, MIN nu va alege nodul 4. Cu toate că se cunoaşte că, valorile nodurilor 10 şi 11 sînt obţinute, valoarea nodului 4 este cel puţin 5. Aşa că nu poate fi mai avantajos pentru MIN să aleagă nodul 4 în detrimentul nodului 3. Se deduce că MIN va alege nodul 3

fără să mai examineze nodul 12. În acest mod valoarea ce determină costul nodului 1 este 5 şi ca urmare valoarea lui S trebuie să fie cel puţin 5. Considerînd în acest moment că valoarea evaluată 4, este obţinută pentru nodul 5, prin examinarea valorilor tuturor nodurilor fiu ale sale, şi cum selecţia nodului 2 nu va aduce pentru MAX avantaje, nu mai este necesară inspectarea nodului 6 şi a celor care sînt fii ai săi. Metoda de segmentare a numărului de noduri ce sînt cercetate este numită şi metoda alfa-beta (alfa-beta method). Pentru căutarea sistematică a arborilor adînci, metoda este simplă de înţeles dacă se formalizează după cum urmează. Se notează marginea inferioară şi superioară pentru fiecare nod cu alfa, respectiv beta. În starea iniţială, pentru nodul S alfa şi beta sînt $-\infty$, respectiv $+\infty$. Pentru aceasta se va scrie $\alpha\beta(S) = (-\infty, \infty)$. Dacă o valoarea x este mai mică decît alfa sau mai mare decît beta se spune că x depăşeşte domeniul (α, β). Cînd valoarea nodului 3 este determinată $\alpha\beta(1) = (-\infty, 5)$. Dacă se cunoaşte că valorile nodurilor fii ale nodului 1 depăşesc gama $(-\infty, 5)$, condiţia pentru întreruperea căutării este îndeplinită. Cînd nodul 11 a fost explorat $\alpha\beta(4) = (5, \beta)$, deci în afara gamei $(-\infty, 5)$, deci condiţia de întreruperea căutării este din nou satisfăcută şi nu va fi necesară căutarea în continuare a fiilor lui 4. La acest moment $\alpha\beta(1) = (5, 5)$ şi $\alpha\beta(5) = (5, \infty)$. Cînd căutarea continuă valoarea nodului 5 este găsită ca fiind 4, în afara gamei $(5, \infty)$ şi se întrerupe căutarea nodului 2 şi a celor ce pornesc din el.

Prima întrerupere a căutării apare datorită faptului că nodul 4 este cel puţin egal cu valoarea lui beta şi a părinţilor săi, a doua întrucît valoarea nodului 2 este cel puţin egală cu valoarea lui alfa. Procedeul este numit beta-cut, respectiv alfa-cut. Metoda de creştere a eficienţei căutării utilizînd marginile inferioare şi superioare este numită metoda alfa-beta. Pentru a determina eficienţa căutării se calculează numărul maxim al nodurilor explorate în cel mai bun caz. Considerînd momentul în care s-a găsit valoarea k a primului nod fiu de la un nod MAX, nodul n. $\alpha\beta(n) = (k, \infty)$. Dacă beta al nodului părinte m al lui n este mai mic sau egal cu k, nu este necesară evaluarea celorlalte noduri fiu ale lui n. Dacă nodul m nu este primul nod fiu al părintelui său, valoarea beta este deja dată şi nu poate fi ∞. Concluzii similare pot fi trase pentru nodurile MIN. În consecinţă, în cel mai bun caz, dacă nodul m nu este primul nod al unui nod părinte, nu este necesară examinarea nici unui nod fiu decît a primului nod al lui m.

Nodurile terminale sînt reprezentate prin secvenţa de arce ce le leagă. De exemplu nodul 12 din fig. 7.10 poate fi exprimat prin secvenţa (1, 2, 3), adică este atins urmărind primul arc al lui S, al doilea de la nodul 1 şi al treilea de la nodul 4. Ca urmare nodurile terminale ale arborelui de adîncime k se exprimă prin k-uplul $(e_1, e_2, \ldots, e_k)$. Nodurile ce nu necesită examinarea sînt cele pentru care $e_1 \# 1$, $e_{i+1} \# 1$. În consecinţă nodurile care au fost examinate în cel mai bun caz, fie sînt noduri numere pare care au în secvenţă 1, fie sînt noduri numere impare care au în secvenţa 1. Noţiunea de par sau impar se referă la numărul nodului în adîncimea arborelui.

▼ *Fig. 7.11. Arbore ŞI/SAU pentru jocuri*



Dacă se presupune că numărul arcelor posibile de la noduri neterminale este m, atunci numărul nodurilor pare $N_0(k, m)$ şi numărul nodurilor impare $N_e(k, m)$ sînt date prin următoarele formule:

dacă k este par $N_0(k, m) = m^{(k-1)/2}$, $N_e(k, m) = m^{(k+1)/2}$

dacă k este impar $N_0(k, m) = m^{k/2}$, $N_e(k, m) = m^{k/2}$

Nodul (1, 1,..., 1) este numărat pentru ambele $N_0(k, m)$ şi $N_e(k, m)$, aşa că $N_{total}(k, m)$ este $N_0(k, m) + N_e(k, m) - 1$. Cu aceste precizări se obţine

pentru k par $N_{total}(k, m) = m^{(k-1)/2} + m^{(k+1)/2} - 1$

pentru k impar $N_{total}(k, m) = 2m^{k/2} - 1$.

În cel mai bun caz, numărul nodurilor ce trebuiesc examinate prin metoda alfa-beta este aproximativ acelaşi cu numărul nodurilor ce trebuie căutat prin metoda minimax într-un arbore de adîncime jumătate. În cazul cel mai defavorabil toate nodurile vor trebui examinate. Uzual situaţia este cuprinsă între două extreme, valoarea reală fiind dată prin inegalitatea

$$m^{k/2} < N(k, m) <= m^k$$

7.4.4. Utilizarea metodei de căutare în grafuri ŞI/SAU

Un arbore specific pentru jocuri este reprezentat ca un graf ŞI/SAU. De exemplu arborele din fig. 7.9 reprezentat ca un graf este arătat în fig. 7.11. Nodurile fiu ale nodurilor MAX corespund la noduri SAU, cele ale nodurilor MIN la noduri ŞI (nodul de start este văzut ca un nod ŞI). Ca şi la arborii ŞI/SAU nodurile terminale au valoarea evaluată dată. Costul arcelor este întotdeauna 0. Utilizînd metoda minimax, cea mai bună mutare este determinată prin următoarele metode:

- dacă nodurile fiu ale unui nod dat sînt noduri ŞI valoarea nodului este dată de cea mai mică valoare evaluată a fiilor săi;
- dacă nodurile fiu sînt noduri SAU valoarea este dată de valoarea maximă dintre valorile nodurilor fiu ale sale.

În cazul jocurilor pentru care obiectivul este cel de a face valoarea nodului de start maximă, valorile evaluate sînt în opoziţie faţă de cost, adică cea mai mare valoarea evaluată este cea mai bună. În consecinţă, metoda de calcul a valorilor în diverse noduri ale grafului este diferită de cea utilizată în metodele descrise anterior pentru arbori ŞI/SAU. Valoarea evaluată a nodului terminal a soluţiei grafului este însăşi valoarea nodului terminal. Valoarea într-un nod unde fii sînt noduri ŞI este valoarea minimă din valorile asociate nodurilor fiu. Valoarea unui nod unde fii sînt noduri SAU este aceeaşi cu valoarea nodului fiu. Dacă se scrie valoarea nodului n ca $f(n)$, $f(A) = 2$ şi $f(S) = 2$. Alte soluţii pot apare din valorile celorlalte noduri. În cazul de faţă soluţia este 1. Metoda descrisă în capitolul anterior referitoare la căutarea în grafuri ŞI/SAU poate fi aplicată şi la arbori de jocuri. Totuşi, $h'(f')$ nu este dată de aceeaşi formulă exceptînd nodurile terminale. Pentru a obţine soluţia optimă condiţia $f'(n) >= f(n)$ este îndeplinită pentru toate nodurile (cînd se urmăreşte obţinerea soluţiei de cost minim $h'(n) <= h(n)$ pentru toate nodurile). În consecinţă dacă valorile punctelor terminale nu sînt date, valorile inferenţiate ale arborilor candidat sînt infinit de mari. Presupunînd că, la expandarea nodurilor neterminale ale grafului candidat, nodurile fiu sînt noduri ŞI ele se expandează la acelaşi moment. Dacă se notează primul nod fiu cu n_1 (nod ŞI) al lui n_i, se poate vedea că la momentul în care valoarea lui n_i este găsită, $f(n) <= f(n_1)$. Ca urmare se poate lua $f'(n) <= f(n_1)$. Aşa se poate calcula f' şi reactualiza în timpul procesului de căutare. Asemănător cu metoda de căutare optimală descrisă în capitolul precedent, principiul acestei metode este bazat pe faptul că un graf candidat adus din open_list va fi expandat după cea mai mare valoare inferenţiată.

În arbori ŞI/SAU pentru jocuri găsirea soluţiei necesită nu numai nodul terminal care dă valoarea soluţiei arborelui ci şi drumul urmat pentru ca pornind de la nodul de start acesta să fie atins. Pentru transpunerea algoritmului, prin convenţie se va nota cu EXPL nodurile atinse ce nu au starea SOLVED. În acest context starea nodului n este dată de tripletul (n, s, v), în care s indică faptul că nodul n este SOLVED sau EXPL, v dă valoarea lui $f(n)$ dacă n este SOLVED sau $f'(n)$ dacă el este EXPL. Starea nodului n este pusă în open_list printr-o procedură pe care am notat-o putopen (n, s, v). De asemenea se va scrie pentru nodul părinte parent(n), nodul terminal n ca terminal (n) şi pentru a detecta un nod n ca fiind ŞI/SAU s-a utilizat funcţia type(n). Se dă în pseudocod algoritmul de căutare în astfel de situaţii.

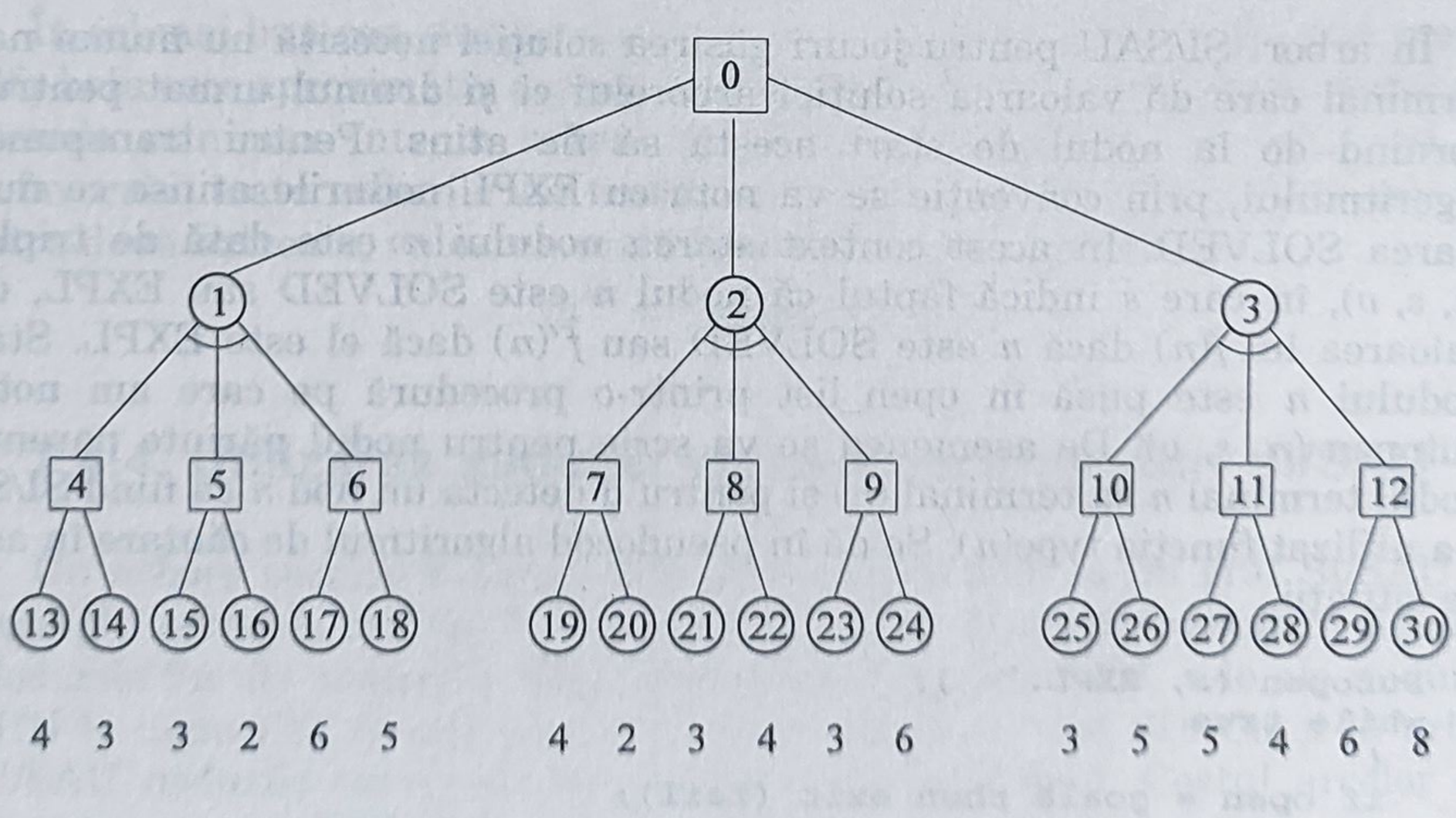
```
Putopen (S, EXPL.    );
while true
  {
  if open = goală then exit (fail);
  n = first (open);
  if (n = S & solved (n)) then exit (succes);
```

```
    remove (n, open);
/*1*/  if ((s = EXPL) & (type(n) = AND) & not_terminal (n)) then
         pentru toate nodurile fiu n_i ale lui n putopen (n, EXPL,
         f'(n));
/*2*/  if ((s = EXPL) & (type(n) = OR) & not_terminal (n))
       {next_n = next_fiu ( )
       if (next_n = SOLVED)
         putopen (n, SOLVED, f'(n));
       else putopen (n, EXPL, f'(n));}
/*3*/  if ((s = EXPL) & terminal (n))
       putopen (n, SOLVED, min (f'(n), f(n));
/*4*/  if ((s = SOLVED) & (type(n) = AND))
       putopen (parent(n), EXPL, f'(n));
/*5*/  if ((s = SOLVED) & (type(n) = OR))
       {m = parent(n);
       mută toate nodurile fiu ale lui m în open_list
       putopen (n, SOLVED, f'(n));}
  }
```

Pentru exemplificarea metodei se consideră graful din fig. 7.12. Aşa cum se vede din rezultatele indicate în fig. 7.13 metoda alfa-beta este dependentă de ordinea nodurilor. În cazul de faţă dacă nodul 3 ar fi în locul nodului 1 soluţia optimă este găsită mult mai rapid. În algoritm au fost notate cele 5 situaţii ce sînt ilustrate şi în fig. 7.13 pentru a uşura cititorul în înţelegerea algoritmului.

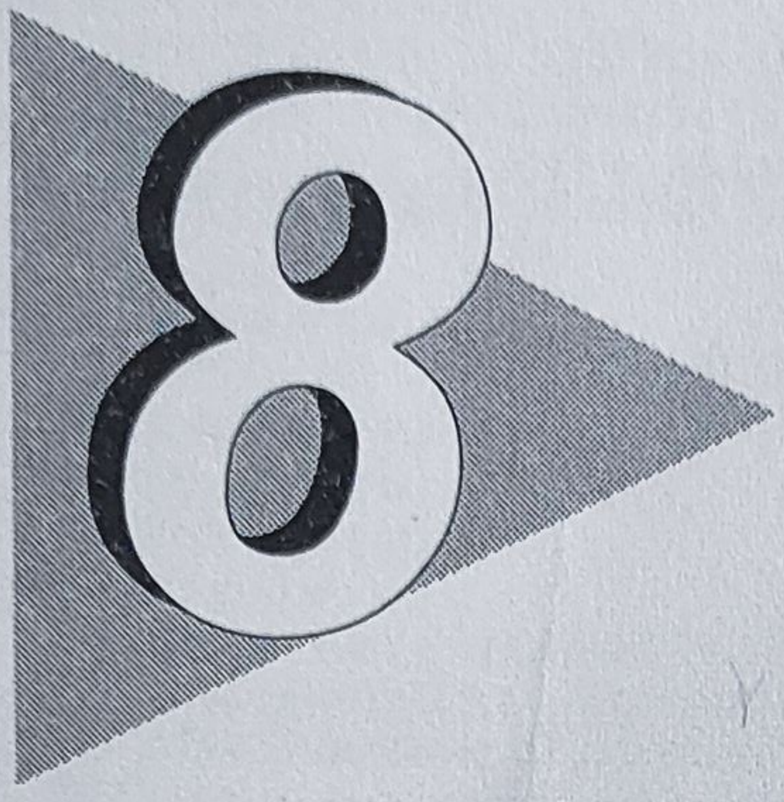
Spre deosebire de căutarea optimală obişnuită care explorează mai întîi cele mai promiţătoare noduri şi a cărei eficienţă nu este afectată de aranjarea nodurilor, metoda alfa-beta este dependentă de aceasta. În al doilea rînd memoria necesară pentru algoritmul alfa-beta este mult mai redusă. Alegerea uneia din cele două metode este dependentă de performanţele dorite din punctul de vedere al vitezei de calcul şi al memoriei necesare.

▼ ***Fig. 7.12. Căutarea soluiei optime în arbore de joc ŞI/SAU***



▼ ***Fig. 7.13. Conţinutul open_list şi procedura utilizată***

Pas	Proc	open_list în care S este SOLVED, E este EXPL
1		$(0, E, \infty)$
2	1	$(1, E, \infty)$, $(2, E, \infty)$, $(3, E, \infty)$
3	2	$(4, E, \infty)$, $(2, E, \infty)$, $(3, E, \infty)$
4	1	$(13, E, \infty)$, $(14, E, \infty)$, $(2, E, \infty)$, $(3, E, \infty)$
5	3	$(14, E, \infty)$, $(2, E, \infty)$, $(3, E, \infty)$, $(13, S, 4)$
6	3	$(2, E, \infty)$, $(3, E, \infty)$, $(13, S, 4)$, $(14, S, 4)$
7	2	$(7, E, \infty)$, $(3, E, \infty)$, $(13, S, 4)$, $(14, S, 3)$
8	1	$(19, E. \infty)$, $(20, E. \infty)$, $(3, E, \infty)$, $(13, S, 4)$, $(14, S, 3)$
9	3	$(20, E, \infty)$, $(3, E, \infty)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$
10	3	$(3, E, \infty)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(20, S, 2)$
11	2	$(10, E, \infty)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(20, S, 2)$
12	1	$(25, E, \infty)$, $(26, E, \infty)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(20, S, 2)$
13	3	$(26, E, \infty)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(25, S, 3)$, $(20, S, 2)$
14	3	$(26, S, 5)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(25, S, 3)$, $(20, S, 2)$
15	5	$(10, S, 5)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(20, S, 2)$
16	4	$(3, E, 5)$, $(13, S, 4)$, $(19, S, 4)$, $(14, S, 3)$, $(20, S, 2)$
28	5	$(S, S, 5)$, $(13, S, 4)$, $(19, S, 4)$, $(4, S, 3)$, $(20, S, 2)$



CONTROLUL REZOLVĂRII PROBLEMELOR

În modul de rezolvare a problemelor s-a folosit reprezentarea în spaţiul stărilor. Cunoaşterea utilizată pentru realizarea unei căutări eficiente este dată sub forma costului stărilor specificate, a costului obiectivului, sau a costului de la starea iniţială la o stare specificată. Cînd problema devine complexă este dificil să se reducă aceasta la o simplă problemă de căutare. Cum se determină strategia de rezolvare a problemei este o chestiunea importantă. În acest capitol se tratează metodele de rezolvare a problemelor complicate utilizînd planuri de rezolvare.

8.1. General problem solver (GPS)

GPS este un model al modului de rezolvare a problemelor de către oameni ce a fost introdus de filozofii A. Newell, J. C. Shaw, şi H. A. Simion. Metoda utilizează un schelet neparticular pentru o problemă dată, însă pentru rezolvarea unei probleme arbitrare utilizează cunoaşterea dată pentru problemă. Întrucît scopul este de a introduce cititorul în metoda GPS nu se tratează decît algoritmul specific GPS. Formalizarea GPS necesită introducerea conceptului de procedură recursivă, procedura care se utilizează pe ea însăşi în propria definire. Procedura ce se utilizează pe ea însăşi în conversia stării *S* într-o altă stare *G* este:

```
Procedura recursivă GPS (S, G)
If S    include G then return (S); /* dacă starea S satisface starea
        G atunci obiectivul este atins */
Caută   diferenţa între G şi S. Depune aceasta într-o listă numită
        difference_list ordonată în ordinea importanţei;
while true
     {
     if stare (difference_list) = vidă then return (eşec);
     d = first (difference_list);
     elimină   (d, difference_list); /* elimină articolul extras din
               difference_list */
     Găseşte   toţi operatorii care reduc diferenţa d şi îi plasează
               în operator_list;
     while (stare (operator_list) # vidă)
```

```
{
op = first (difference_list);
elimină (op, operator_list);
pc = precondiţiile operatorului;
S1 = GPS (S, pc); /* S1 este o nouă stare după aplicarea
     procedurii GPS (S, pc) */
If (S1 = eşec) then break;
S2 = op (S1); /* aplică op asupra lui S1 */
S3 = GPS (S2, G);
If (S3 = eşec) then break;
}
}
return (S3).
```

Dacă în problemă se cere schimbarea stării iniţiale S_0 în starea finală G_0, se rezolvă prin apelarea procedurii recursive GPS (S_0, G_0). Pentru aceasta GPS divide obiectivul de atingerea lui G într-un subobiectiv al lanţului de la starea S la starea S_1 satisfăcînd precondiţiile pc, apoi subscopul de atingerea stării S_2 prin aplicarea operatorilor asupra stării S_1, şi aşa mai departe.

8.2. Planuri

Cînd procedura pentru rezolvarea unei probleme date devine foarte complicată o metodă eficientă este cea de stabilire a unui plan general pentru obţinerea soluţiei. Aşa cum s-a menţionat mai sus, o metodă este bazată pe reducerea celei mai importante diferenţe între starea curentă şi starea obiectiv. În acest paragraf se tratează modul în care se formalizează planul pentru rezolvarea problemelor de aşezare a cutiilor. Mai întîi se descriu metodele de reprezentare a problemelor şi metodele de căutare pentru obţinerea soluţiilor, modul în care acestea sînt îmbunătăţite prin utilizarea planului.

8.2.1. Reprezentarea problemelor şi metode de soluţionare de bază

Rezolvarea unei probleme constă în modificarea unei stări date numită şi stare iniţială într-o stare finală numită şi stare obiectiv. O metodă de reprezentare a stării este şi calculul cu predicate, situaţie în care sînt definiţi o serie de operatori, iar pentru fiecare operator modul în care starea se modifică şi care sînt precondiţiile pentru ca aceasta să fie aplicabil. Ca exemplul se consideră problema din fig. 8.1.

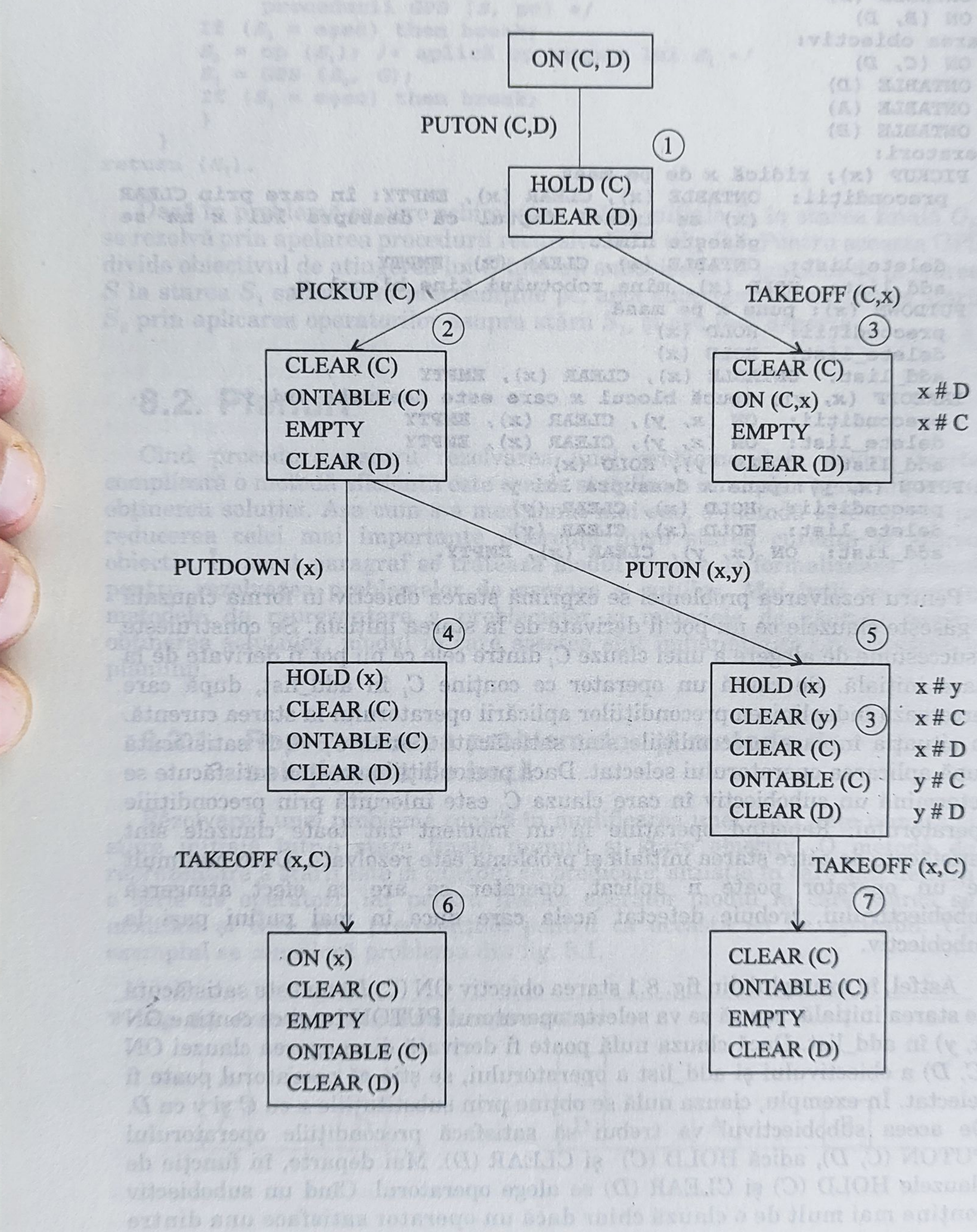
▼ ***Fig. 8.1. Problema simplă de aranjare cuburi***

```
Starea inițială:
   EMPTY; brațul robotului nu ține nimic
   ONTABLE (C); blocul C se găsește pe masă
   ON (A, C); A este deasupra lui C
   ONTABLE (D)
   ON (B, D)
Starea obiectiv:
   ON (C, D)
   ONTABLE (D)
   ONTABLE (A)
   ONTABLE (B)
Operatori:
   PICKUP (x); ridică x de pe masă
     precondiții:  ONTABLE (x), CLEAR (x), EMPTY: în care prin CLEAR
                   (x) se indică faptul că deasupra lui x nu se
                   găsește nimic
     delete_list:  ONTABLE (x), CLEAR (x), EMPTY
     add_list:  HOLD (x), mîna robotului ține blocul x.
   PUTDOWN (x): pune x pe masă
     precondiții:  HOLD (x)
     delete_list:  HOLD (x)
     add_list:  ONTABLE (x), CLEAR (x), EMPTY
   TAKEOFF (x, y); apucă blocul x care este deasupra lui y
     precondiții:  ON (x, y), CLEAR (x), EMPTY
     delete_list:  ON (x, y), CLEAR (x), EMPTY
     add_list:  CLEAR (y), HOLD (x)
   PUTON (x, y): pune x deasupra lui y
     precondiții:  HOLD (x), CLEAR (y)
     delete_list:  HOLD (x), CLEAR (y)
     add_list:  ON (x, y), CLEAR (x), EMPTY.
```

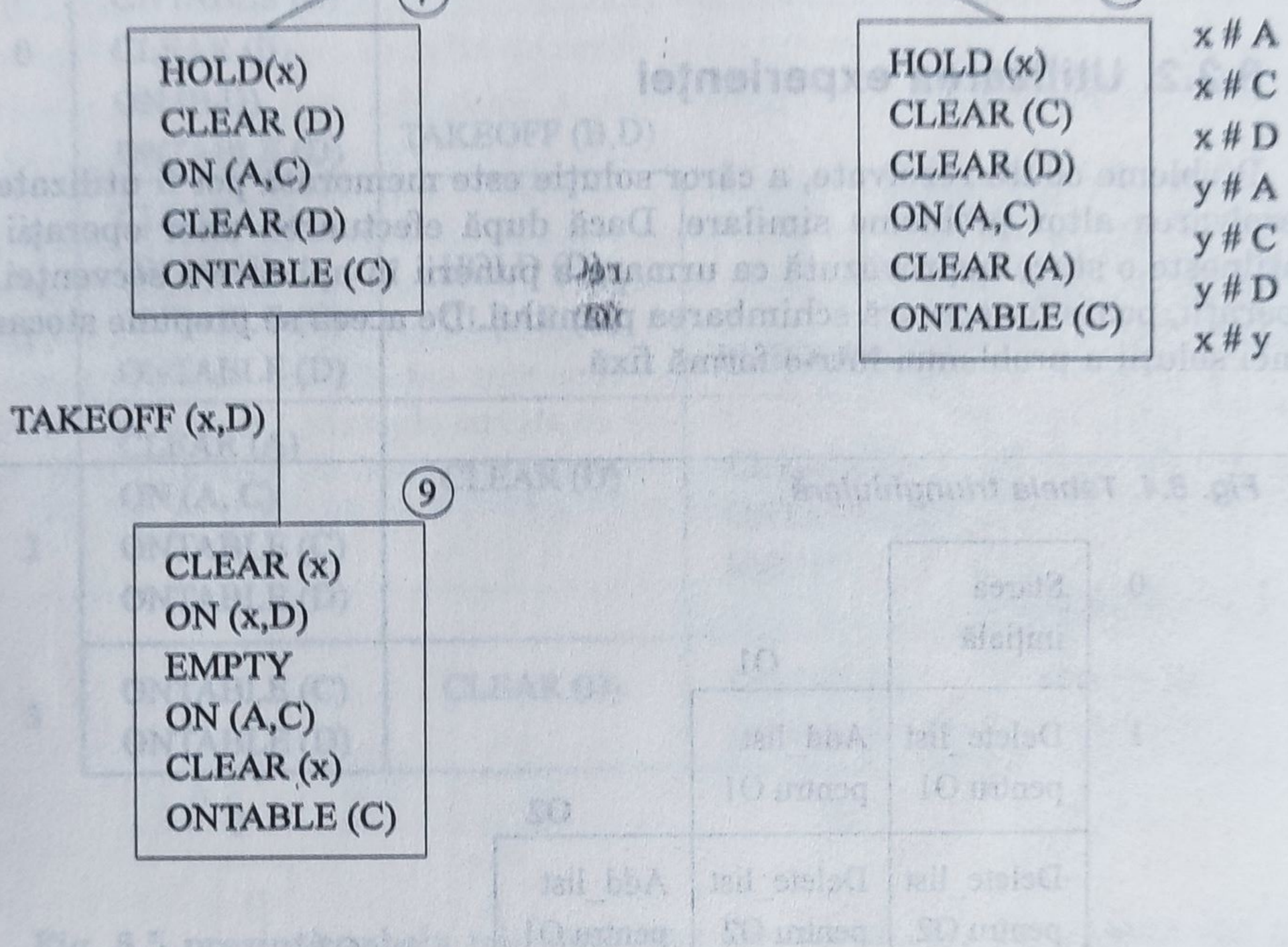
Pentru rezolvarea problemei se exprimă starea obiectiv în forma clauzală și găsește clauzele ce nu pot fi derivate de la starea inițială. Se construiește o succesiune de alegere a unei clauze C_i dintre cele ce nu pot fi derivate de la stare inițială. Se caută un operator ce conține C_i în add_list, după care cercetează îndeplinirea precondițiilor aplicării operatorului la starea curentă. În situația în care precondițiile sînt satisfăcute clauza C_i va fi satisfăcută după aplicarea operatorului selectat. Dacă precondițiile nu sînt satisfăcute se determină un subobiectiv în care clauza C_i este înlocuită prin precondițiile operatorului. Repetînd operațiile la un moment dat toate clauzele sînt satisfăcute de către starea inițială și problema este rezolvată. Dacă mai mult de un operator poate fi aplicat, operator ce are ca efect atingerea subobiectivului, trebuie detectat acela care duce în mai puțini pași la subobiectiv.

Astfel, în exemplul din fig. 8.1 starea obiectiv ON (*C, D*) nu este satisfăcută de starea inițială, așa că se va selecta operatorul PUTON (*x, y*) ce conține ON (*x, y*) în add_list. Dacă clauza nulă poate fi derivată din negarea clauzei ON (*C, D*) a obiectivului și add_list a operatorului, se știe că operatorul poate fi selectat. În exemplu, clauza nulă se obține prin substituțiile *x* cu *C* și *y* cu *D*. De aceea subobiectivul va trebui să satisfacă precondițiile operatorului PUTON (*C, D*), adică HOLD (*C*) și CLEAR (*D*). Mai departe, în funcție de clauzele HOLD (*C*) și CLEAR (*D*) se alege operatorul. Cînd un subobiectiv conține mai mult de o clauză chiar dacă un operator satisface una dintre

▼ ***Fig. 8.2. Prima parte a procesului de rezolvare din fig. 8.1***



▼ *Fig. 8.3. Continuarea procesului din starea (6)*



clauze nu va fi selectat dacă alte clauze sînt inconsistente cu delete_list sau add_list a operatorului. De exemplu operatorul PUTDOWN (*D*) care are CLEAR (*D*) în add_list conţine şi EMPTY în add_list. Cum clauza este inconsistentă cu obiectivul HOLD(*C*) operatorul nu va fi selectat. Prima parte a soluţiei acestei probleme este dată în fig. 8.2. Restricţiile asupra lui *x* sînt aplicate în starea (3). Comparînd stările (2) şi (3), (2) conţine mai puţine clauze ce nu sînt îndeplinite de starea iniţială aşa că este expandată. Clauzele din (2) care nu sînt îndeplinite de starea iniţială sînt CLEAR (*C*) şi CLEAR (*D*). Fiecare dintre operatori pot avea o prioritate dată. Clauzele din starea (6) în afară de CLEAR (*D*) sînt satisfăcute de starea iniţială dacă se face substituţia lui *x* cu *A*. În dreapta stării (5) sînt indicate restricţiile variabilelor.

Dacă se face substituţia lui x cu D în (5) se obţine o contradicţie. Substituţia lui x cu A este găsită în starea (7). Această substituţie extinsă la (5) determină substituţia lui y cu B. Ambele stări (6) şi (7) sînt părţi ale soluţiei. Dacă se urmează calea dată de (7) starea finală este ON (C, D), ON (A, B) este doar o soluţie parţială. În fig. 8.3 s-a prezentat numai continuarea din starea (6).

Se observă că starea (8) este nerealizabilă, starea (9) este satisfăcută de starea iniţială dacă se face substituţia lui x cu B. Ca urmare soluţia va fi:

TAKEOFF (B, D), PUTDOWN (B), TAKEOFF (A, C), PUTDOWN (A), PICKUP (C), PUTON (C, D).

Se remarcă faptul că problema simplă cum este cea din figură necesită un număr destul de mare de căutări şi problemele complicate nu pot fi rezolvate prin metodele elementare date aici.

8.2.2. Utilizarea experienţei

Probleme odată rezolvate, a căror soluţie este memorată pot fi utilizate la rezolvarea altor probleme similare. Dacă după efectuarea unor operaţii se întîlneşte o stare neprevăzută ca urmare a punerii în aplicare a secvenţei de operaţii, poate fi necesară schimbarea planului. De aceea se propune stocarea unei soluţii a problemei într-o formă fixă.

▼ ***Fig. 8.4. Tabela triunghiulară***

		O1	O2	O3	On
0	Starea iniţială				
1	Delete_list pentru O1	Add_list pentru O1			
	Delete_list pentru O2	Delete_list pentru O2	Add_list pentru O1		
					Add_list pentru On

Forma fixă este numită tabelă triunghiulară şi este prezentată în fig. 8.4. Starea iniţială este introdusă în rîndul 0 coloana 0. Add_list obţinută prin aplicarea operatorului O_1 asupra acesteia este introdusă în rîndul 1 coloana 1. În rîndul 1 coloana 0 se introduce rezultatul eliminării din delete_list a lui O_1 de la starea iniţială. Cu alte cuvinte starea după aplicarea operatorului O_1 este arătată în coloanele rîndului 1. Rezultatul aplicării operatorului O_2 la această stare este prezentat în rîndul 2. Starea din rîndul m după aplicarea operatorului O_m satisface stare obiectiv.

▼ ***Fig. 8.5. Parte a tabelei triunghiulare pentru exemplul de aşezare a cutiilor***

	0	1	2	3
0	EMPTY CLEAR (A) ON (A, C) ONTABLE (C) CLEAR (B) ON (B,D) ONTABLE (D)	TAKEOFF (B,D)		
1	CLEAR (A) ON (A, C) ONTABLE (C) ONTABLE (D)	HOLD (D) CLEAR (D)	PUTDOWN (B)	
2	CLEAR (A) ON (A, C) ONTABLE (C) ONTABLE (D)	CLEAR (D)	CLEAR (B) ONTABLE (B) EMPTY	TAKEOFF (A,C)
3	ONTABLE (C) ONTABLE (D)	CLEAR (D)	CLEAR (B) ONTABLE (B)	HOLD (A) CLEAR (C)

Fig. 8.5 prezintă tabela triunghiulară pentru soluţia exemplului dat mai sus în primele trei rînduri şi coloane. Dacă se consideră că toate clauzele ce constituie precondiţii ale operatorilor sînt marcate, în ultimul rînd al tabelei clauzele marcate sînt conţinute în starea obiectiv. Tabela de faţă conţine constante, însă reprezentarea poate fi generalizată prin tabela generalizată pentru acest exemplu, în care sînt prezentate numai clauzele marcate.

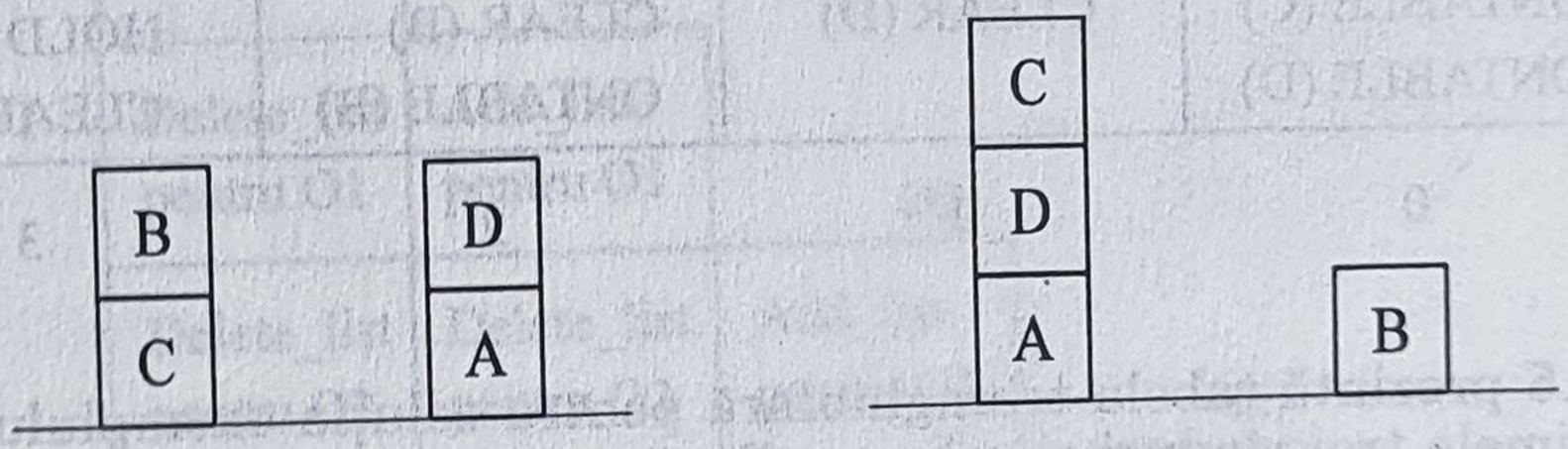
Privită cu atenţie o parte a tabelei triunghiulare conţinută în rîndul i şi la stînga coloanei i, conţine precondiţiile de aplicare a operatorilor care se găsesc de la numărul $i + 1$ la sfîrşitul secvenţei de operatori. De exemplu porţiunea cuprinsă la stînga liniei duble din fig. 8.6 conţine cinci clauze ce sînt condiţiile de aplicare ale celui de al treilea şi al următorilor operatori în secvenţa de operatori. Orice stare ce satisface aceste clauze poate fi convertită în starea obiectiv.

▼ ***Fig. 8.6. Tabela triunghiulară generalizată***

	0	1	2	3	4	5	6
0	CLEAR (y) ON (y,v) EMPTY	TAKEOFF (y,v)					
1		HOLD (y)	PUTDOWN (y)				
2	CLEAR (x) ON (x,u)		EMPTY	TAKEOFF (x,v)			
3				HOLD (x)	PUTDOWN (x)		
4	ONTABLE(u)			CLEAR (u)	EMPTY	PICKUP (u)	
5						HOLD (v)	PUTON (u,v)
6		CLEAR (v)					ON (u,v)

Se consideră un exemplu similar cu cel descris mai sus cu deosebirea că în obiectiv apare atît ON (C, D) cît şi alte obiective şi în plus diferă şi starea iniţială. Structura este prezentată în fig. 8.7. Se observă, în tabela din fig. 8.6 că dacă substituţiile u cu C, v cu D şi x cu B sînt executate, porţiunea de sub şi din stînga liniilor duble conţine clauze ce sînt satisfăcute de stare iniţială şi deci starea din al şaselea rînd coincide cu starea obiectiv. Ca urmare, starea obiectiv poate fi atinsă prin secvenţa de operatori după O_3 TAKEOFF (B, C), PUTDOWN (B), PICKUP (C), PUTON (C, D).

▼ ***Fig. 8.7. Problemă de cuburi similară***



Să considerăm acum situaţia cînd robotul întîlneşte o stare neprevăzută la execuţia operaţiilor arătate în fig. 8.6. De exemplu cînd execută ultimul operator PUTON (C, D) el scapă blocul C. Presupunînd că robotul poate găsi blocul C utilizînd o cameră de luat vederi, starea satisface clauzele cuprinse la stînga coloanei 4 şi sub rîndul 4, ceea ce face ca să înceapă cu PICKUP (C). Dacă se dispune de tabele triunghiulare pentru diferite operaţii, este posibilă mutarea dintr-o stare a unei tabele într-o altă stare din altă tabelă. Dacă clauzele din al i-lea rînd al tabelei triunghiulare (incluzînd clauzele nemarcate), satisfac clauzele cuprinse în al i-lea rînd şi la stînga coloanei j din

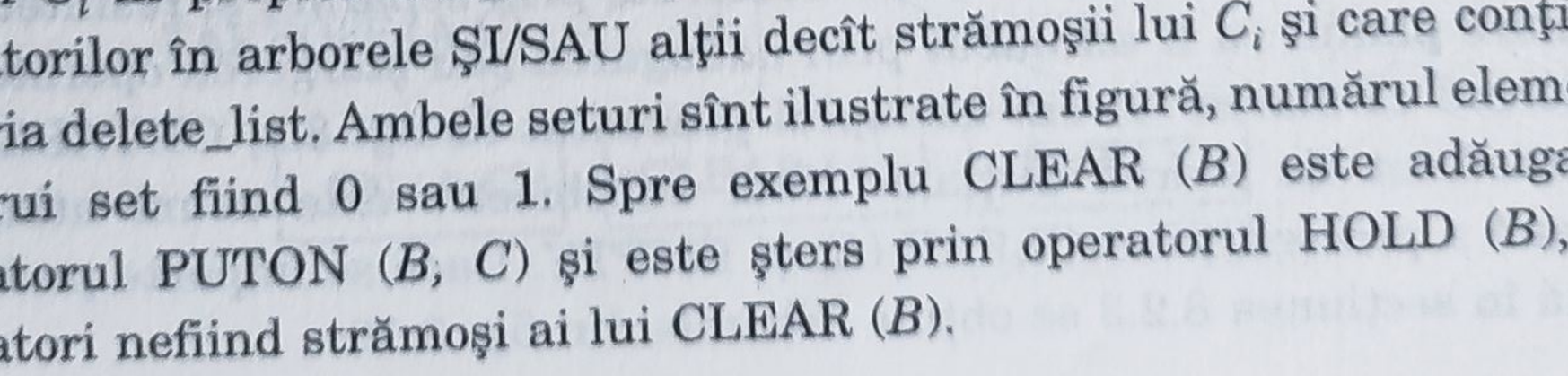
altă tabelă, după aplicarea operatorului O_i se poate muta la al $(j + 1)$-lea operator al altei tabele. Astfel, dacă operaţii comparativ similare sînt executate la mai multe momente de timp, este uzual să se păstreze secvenţa operaţiilor tipice în formă de tabele triunghiulare.

8.2.3. Inferenţe cu mai multe obiective

La problemele anterioare s-a pus problema atingerii unui singur obiectiv, însă pe parcursul procesului atingerea subscopurilor include diferite clauze. Dacă aceste clauze sînt independente soluţia problemei poate fi simplificată prin rezolvarea clauzelor separat. Dacă nu sînt independente, va trebui găsită o secvenţă de operaţii ce ating toate obiectivele în acelaşi timp. În acest paragraf se va explica metoda de rezolvare a diverselor obiective independent, precum şi modul de obţinere a unei soluţii prin considerarea efectelor inferenţelor mutuale între aceste obiective.

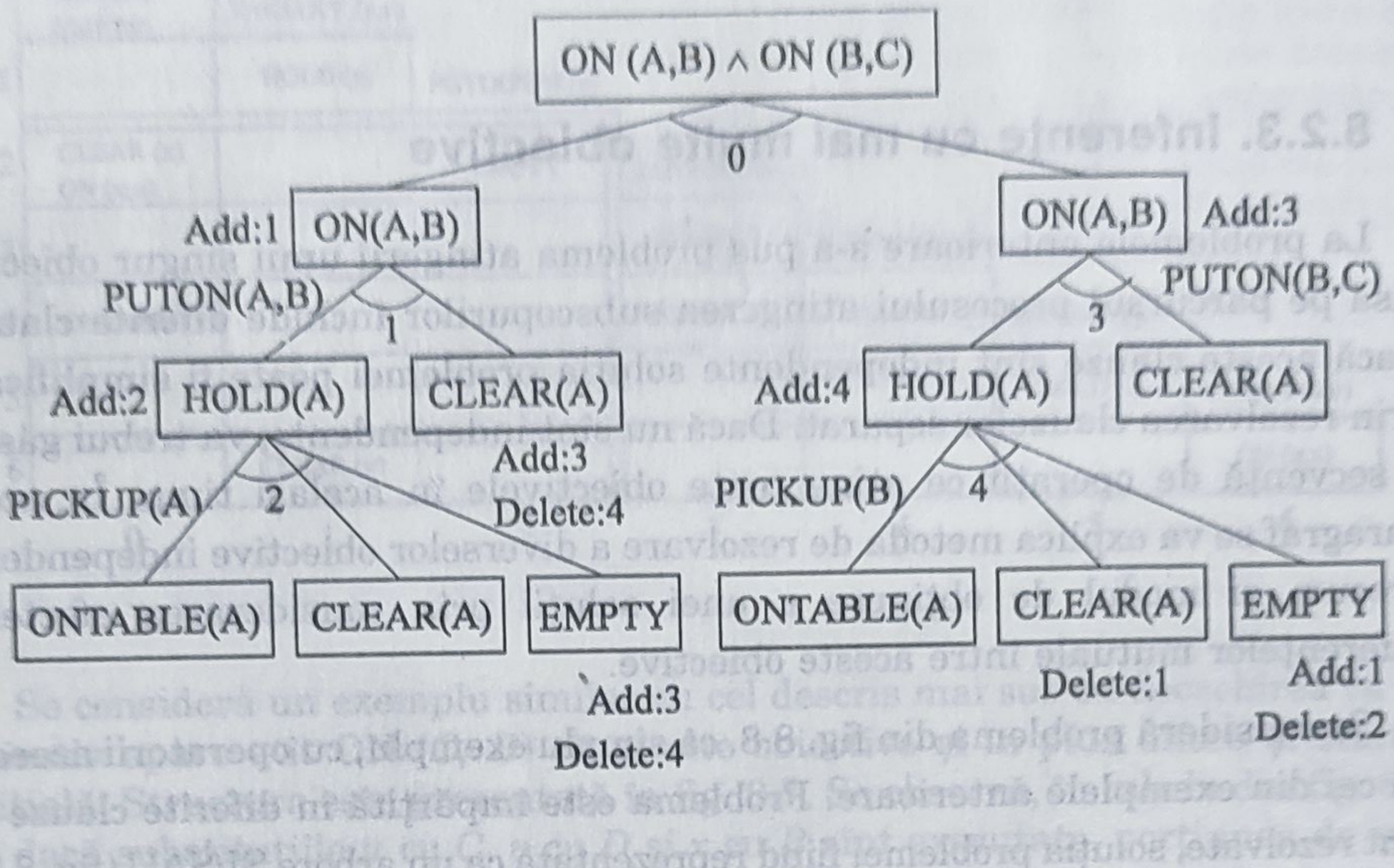
Se consideră problema din fig. 8.8. ca simplu exemplu, cu operatorii aceeaşi cu cei din exemplele anterioare. Problema este împărţită în diferite clauze ce sînt rezolvate, soluţia problemei fiind reprezentată ca un arbore ŞI/SAU (fig. 8.9).

▼ ***Fig. 8.8. Exemplu de construire a turnului***



Operatorii sînt denumiţi prin nume arbitrare (separarea obiectivului iniţial în două obiective este considerată ca operator). Două seturi de operatori sînt definite pentru fiecare obiectiv (add, delete). Operatorii add ai obiectivului C_i sînt setul operatorilor în arborele ŞI/SAU alţii decît strămoşii lui C_i şi care conţin C_i în propria add_list. Operatorii delete ai obiectivuui C_i sînt setul operatorilor în arborele ŞI/SAU alţii decît strămoşii lui C_i şi care conţin C_i în propria delete_list. Ambele seturi sînt ilustrate în figură, numărul elementelor fiecărui set fiind 0 sau 1. Spre exemplu CLEAR (B) este adăugat prin operatorul PUTON (B, C) şi este şters prin operatorul HOLD (B), aceşti operatori nefiind strămoşi ai lui CLEAR (B).

▼ ***Fig. 8.9. Arborele ŞI/SAU al soluţiei cînd diferite obiective sînt considerate independent***

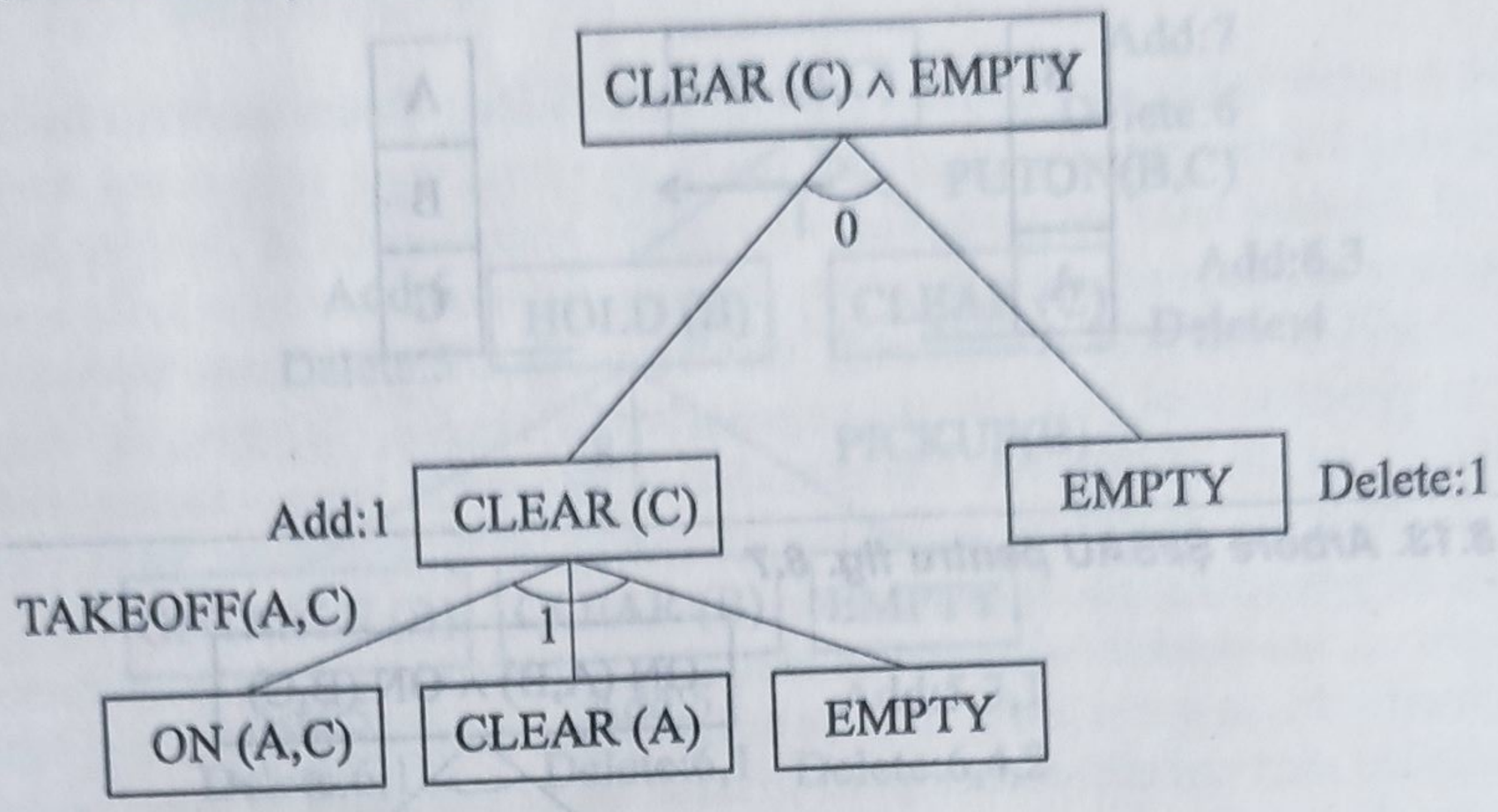


Restricţiile secvenţei de operatori pot fi obţinute din această figură. Pentru ca un operator O_i să fie aplicat, fie operatorii ce îl preced în secvenţă nu sînt incluşi în add_list sau în delete_list a precondiţiilor lui O_i, fie ultimul operator în secvenţă este în add_list. De exemplu precondiţia CLEAR (*B*) a celui de al patrulea operator în secvenţă include 1 în propria delete_list dar nu şi în add_list. Aşa că operatorul 4 nu poate fi aplicat după operatorul 1. Dacă se examinează precondiţia EMPTY a operatorului 2, se vede că nu poate fi aplicat imediat după operatorul 4, însă va cere ca operatorul 3 să fie inserat după operatorul 4. Secvenţa de operatori 4, 3, 2, 1 este de aceea o soluţie candidată. Ea este o soluţie completă întrucît satisface condiţiile de aplicabilitate pentru toţi operatorii.

8.2.4. Schimbarea planului

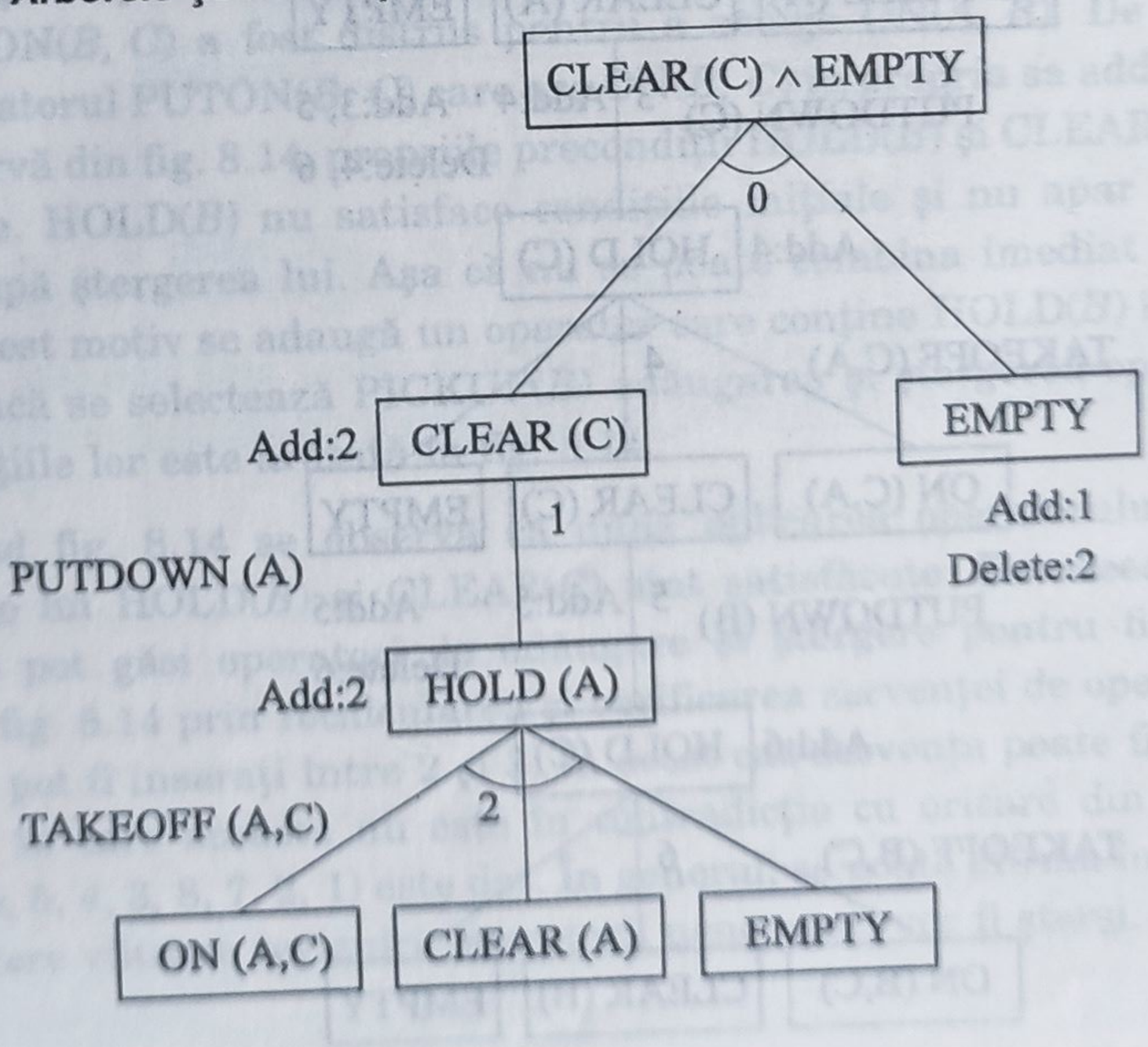
De multe ori problemele nu pot fi rezolvate prin separare şi resintetizare. Ca urmare planul va fi schimbat prin adăugarea sau ştergerea operatorilor. Considerînd că starea (2) din exemplul prezentat în fig. 8.1, ONTABLE (*C*) este satisfăcut de starea iniţială. Noul subobiectiv este acum atingerea simultană a precondiţiilor CLEAR (*C*) şi EMPTY. Dacă se împarte problema descrisă în secţiunea 8.2.3 se obţine arborele din fig. 8.10.

▼ *Fig. 8.10. Arbore ŞI/SAU generat în procesul de separare a problemei*

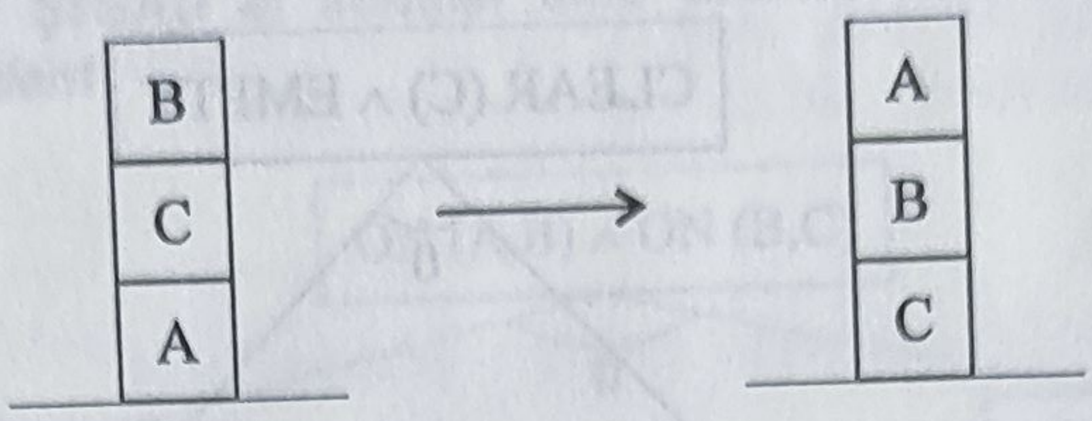


Scopul nu poate fi atins direct întrucât TAKEOFF(*A*, *C*) care este în add_list pentru CLEAR(*C*) este un operator de ştergere pentru EMPTY. Ca urmare, este nevoie de adăugarea unui operator care are EMPTY în propria add_list. Întrucît add_list pentru TAKEOFF(*A*, *C*) include HOLD(*A*) se caută un operator care converteşte HOLD(*A*) în EMPTY. La selecţia operatorului PUTDOWN(*A*) se obţine soluţia din fig. 8.11, metoda devenind mult mai eficientă faţă de găsirea secvenţei de operatori prin căutare ca în paragraful 8.2.1.

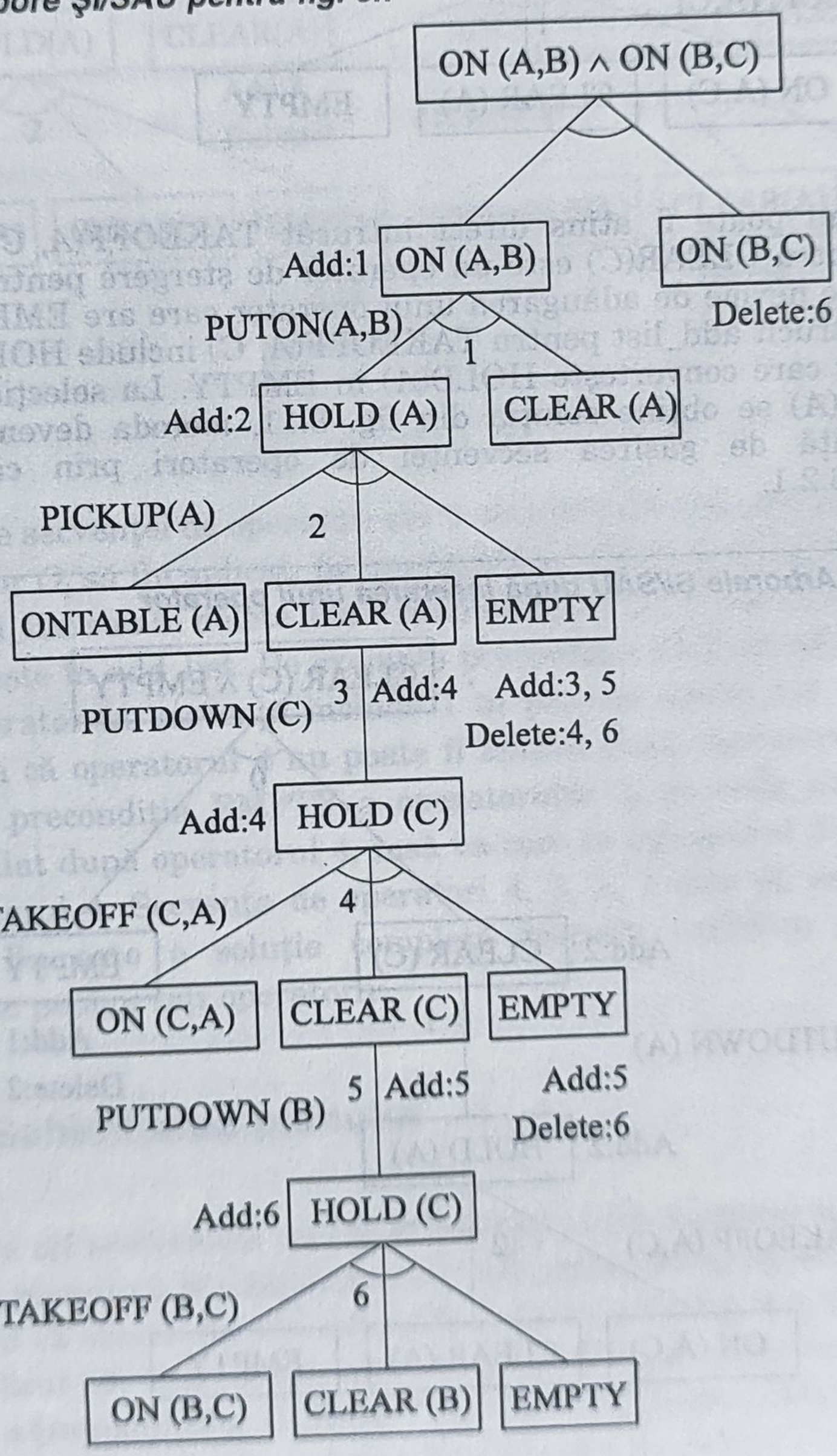
▼ *Fig. 8.11. Arborele ŞI/SAU după inserarea unui operator*



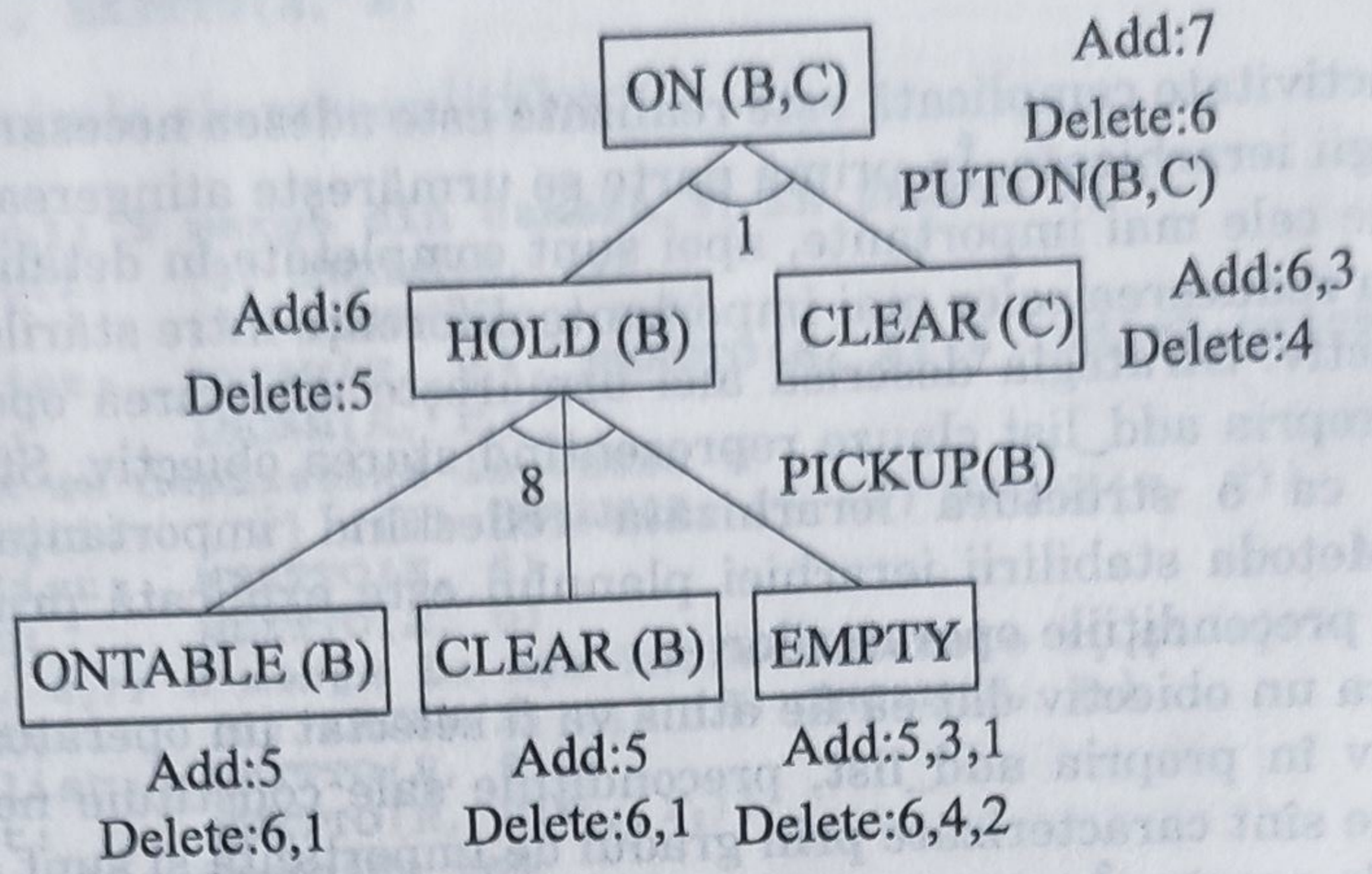
▼ ***Fig. 8.12. Problema ce necesită revizuirea planului***



▼ ***Fig. 8.13. Arbore ŞI/SAU pentru fig. 8.7***



▼ ***Fig. 8.14. Căutarea adăugând operatori***



O serie de probleme nu pot fi rezolvate prin simpla inserare a unui singur operator. În acest caz este necesară construirea unui întreg plan, fără contradicţii, prin urmărirea obiectivului, având un operator final de ştergere şi asigurând adăugarea succesivă a operatorilor care au acest obiectiv în propria add_list.

Se consideră problema din fig. 8.12. Un plan obţinut prin separarea obiectivului şi simpla inserare a operatorilor este arătat în fig. 8.13. Se poate observa că ON(*B, C*) a fost distrus pentru a atinge ON(*A, B*). De aceea se adaugă operatorul PUTON(*B, C*) care are ON(*B, C*) în propria sa add_list. Aşa cum se observă din fig. 8.14, propriile precondiţii HOLD(*B*) şi CLEAR(*C*) devin noi obiective. HOLD(*B*) nu satisface condiţiile iniţiale şi nu apar operatori adăugaţi după ştergerea lui. Aşa că nu se poate combina imediat ca în fig. 8.13. Din acest motiv se adaugă un operator care conţine HOLD(*B*) în propria add_list. Dacă se selectează PICKUP(*B*) adăugarea şi ştergerea operatorilor cu precondiţiile lor este arătată în fig. 8.14.

Urmărind fig. 8.14 se observă că după aplicarea operatorului 3 toate precondiţiile lui HOLD(*B*) şi CLEAR(*C*) sînt satisfăcute. De aceea la acest moment se pot găsi operatori de adăugare şi ştergere pentru fiecare din obiectivele fig. 8.14 prin recalcularea şi verificarea secvenţei de operatori din fig. 8.13 ce pot fi inseraţi între 2 şi 3. În acest caz secvenţa poate fi inserată, şi un plan în care aceasta nu este în contradicţie cu oricare din operatori (secvenţa 6, 5, 4, 3, 8, 7, 2, 1) este dat. În general, se poate afirma în concluzie ca planul cere viitoare revizuiri, operatorii nenecesari vor fi şterşi.

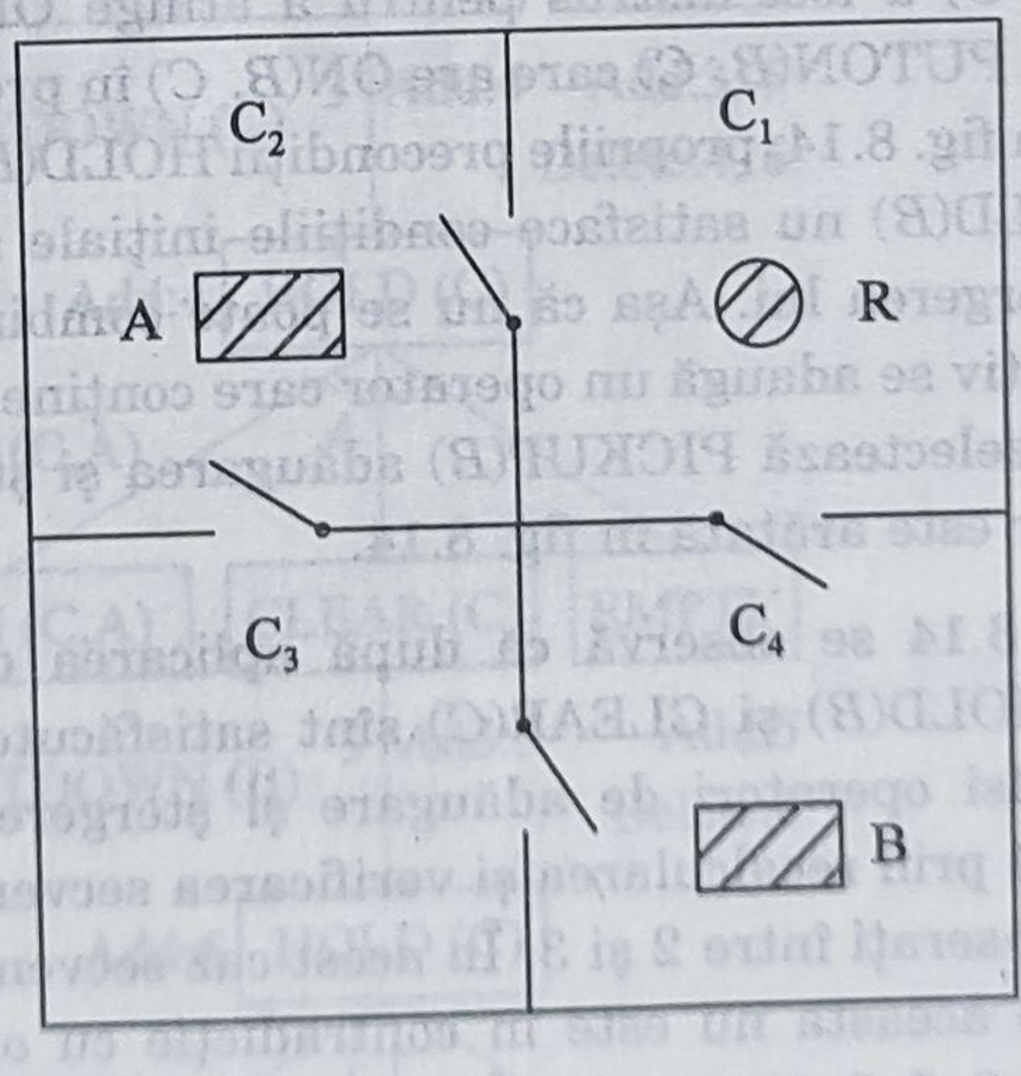
8.2.5. Planuri ierarhizate

Cînd o activitate complicată este realizată este adesea necesară elaborarea unei strategii ierarhizate. În prima parte se urmăreşte atingerea acelor părţi ce par să fie cele mai importante, apoi sunt completate în detaliu strategiile GPS pentru reducerea celor mai importante diferenţe între stările specifice şi starea obiectiv. Strategia descrisă aici urmăreşte selectarea operatorilor ce conţin în propria add_list clauze reprezentînd starea obiectiv. Strategia este prezentată ca o structură ierarhizată reflectând importanţa diferitelor obiective. Metoda stabilirii ierarhiei planului este explicată prin asignarea ierarhiei la precondiţiile operatorilor.

Pentru ca un obiectiv dat să fie atins va fi selectat un operator ce conţine acel obiectiv în propria add_list, precondiţiile sale constituie noi obiective. Precondiţiile sînt caracterizate prin gradul de importanţă şi sunt considerate în funcţie de acesta. În acest mod se aranjază obiectivele într-o ierarhie de importanţe. Ele vor fi utilizate în stabilirea planului de atingere a celui mai important obiectiv. Planul va fi permanent rafinat pentru includerea şi a celor mai puţin importante obiective.

În problema din fig. 8.15 robotul R este localizat în camera C_1 şi trebuie să mute cutiile A şi B localizate în camerele C_2 şi respectiv C_4 una după alta în camera C_3. Între camere există uşi ce sunt specificate prin $U(C_i, C_j)$, indicînd o uşă între camerele C_i şi C_j. Problema se reprezintă prin:

▼ ***Fig. 8.15. Exemplu de aplicare a planurilor ierarhizate***



Starea iniţială:

```
INCAM(R, C1),    INCAM(A, C2),    INCAM(B, C4),    NOTOPEN(U(C1, C2)),
NOTOPEN(U(C2, C3)), NOTOPEN(U(C3, C4)), NOTOPEN(U(C4, C1))
```

Starea obiectiv:

INCAM(A, C_3), NEXTTO(A, B)

Operatori şi nivele ale precondiţiilor:

```
GOTHRU(r1, r2); R merge din camera r1 în camera r2
   precondiţii:  (3) INCAM(R, r1)
                 (2) OPEN(U(r1, r2))
   delete_list:  INCAM(R, r1), NEXTTO(R, $) $ indică orice
   add_list:     INCAM(R, r2)
GOTOB(u); R se deplasează la cutia u
   precondiţii:  ( r) {(3) INCAM(R, r) (3) INCAM(u, r)}
   delete_list:  NEXTTO(R, $)
   add_list:     NEXTTO(R, u)
GOTOU(U(r1, r2)); R merge la uşa între camerele r1 şi r2
   precondiţii:  (3) INCAM(R, r1) (3) INCAM(R, r2)
   delete_list:  NEXTTO(R, $)
   add_list:     NEXTTO(R, U(r1, r2))
OPEN(U(r1, r2)); deschide uşa
   precondiţii:  (1) NEXTTO(R, U(r1, r2))
   delete_list:  NOTOPEN(U(r1, r2))
   add_list:     OPEN(U(r1, r2))
PUSHTHRU(u, r1, r2); robotul mută cutia u din camera r1 în camera r2
   precondiţii:  (3) INCAM(R, r1)
                 (4) INCAM(u, r1)
                 (1) NEXTTO(R, u)
                 (2) OPEN(U(r1, r2))
   delete_list:  INCAM(R, r1), INCAM(u, r1), NEXTTO(u, $)
   add_list:     INCAM(R, r2), INCAM(u, r2), NEXTTO(R, u)
PUSHB(u1, u2); pune cutia 1 după cutia 2
   precondiţii:  ( r)  {(4)  INCAM(u1, r)  (4)  INCAM(u2, r)  (3)
                 INCAM(R, r)}
                 (1) NEXTTO(R, u1))
   delete_list:  nimic
   add_list:     NEXTTO(u1, u2)
```

Axiome

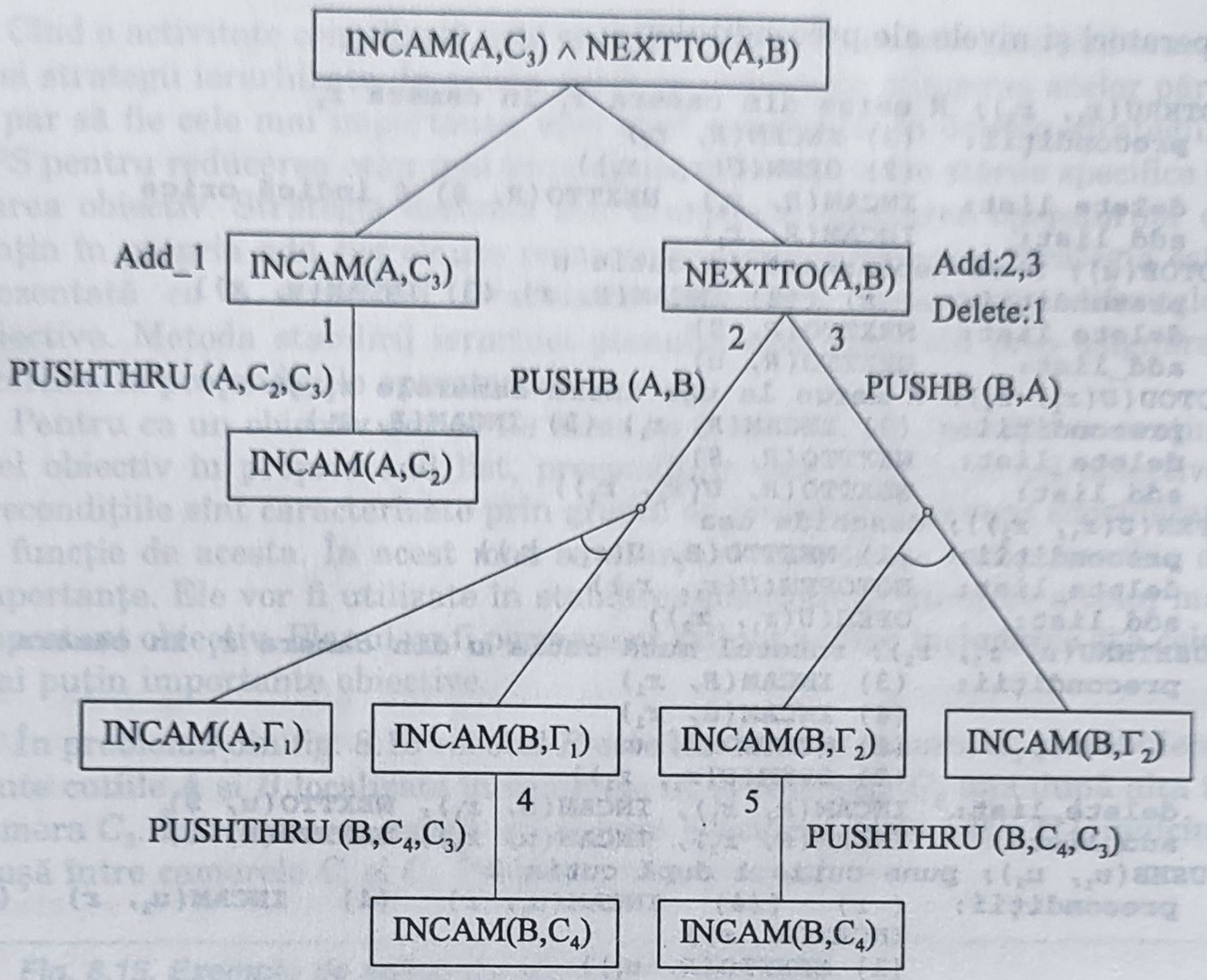
```
OPEN(U(r1, r2)) = OPEN(U(r2, r1))
NEXTTO(u1, u2) = NEXTTO(u2, u1)
```

Rezultatele obţinute prin prima rezolvare în spaţiul abstract al problemei va fi segmentat în mai multe nivele de nivel inferior. Aceste nivele sînt prezentate în continuare:

Nivel 4

Prelucrarea planului care ia în consideraţie numai precondiţiile este prezentată în fig. 8.16. Noduri auxiliare sînt inserate în figură întrucît sînt doi operatori care satisfac NEXTTO(*A*, *B*). Se poate vedea din add_list şi delete_list a operatorului NEXTTO(*A*, *B*) că operatorul 1 trebuie aplicat întotdeauna înaintea operatorilor 2 sau 3. De aceea la momentul în care operatorul 2 sau 3 este aplicat INCAM(*A*, C_3) este true. Din această cauză variabilele r_1 şi r_2 ale precondiţiilor operatorilor 2 şi 3 trebuie să fie C_3. Operatorii 4 şi 5 sînt de aceea determinaţi aşa cum se vede în figură. La nivelul 4 sînt posibile următoarele două soluţii:

▼ ***Fig. 8.16. Arbore ŞI/SAU în spaţiu abstract de nivel 4***



PUSHTHRU(A, C_2, C_3) → PUSHTHRU(B, C_4, C_3)
→ PUSHB(A, B)
→ PUSHB(B, A)

Ambele dintre cele două soluţii sînt corecte, însă va deveni clar la nivelul 1 care dintre acestea necesită mai puţini operatori.

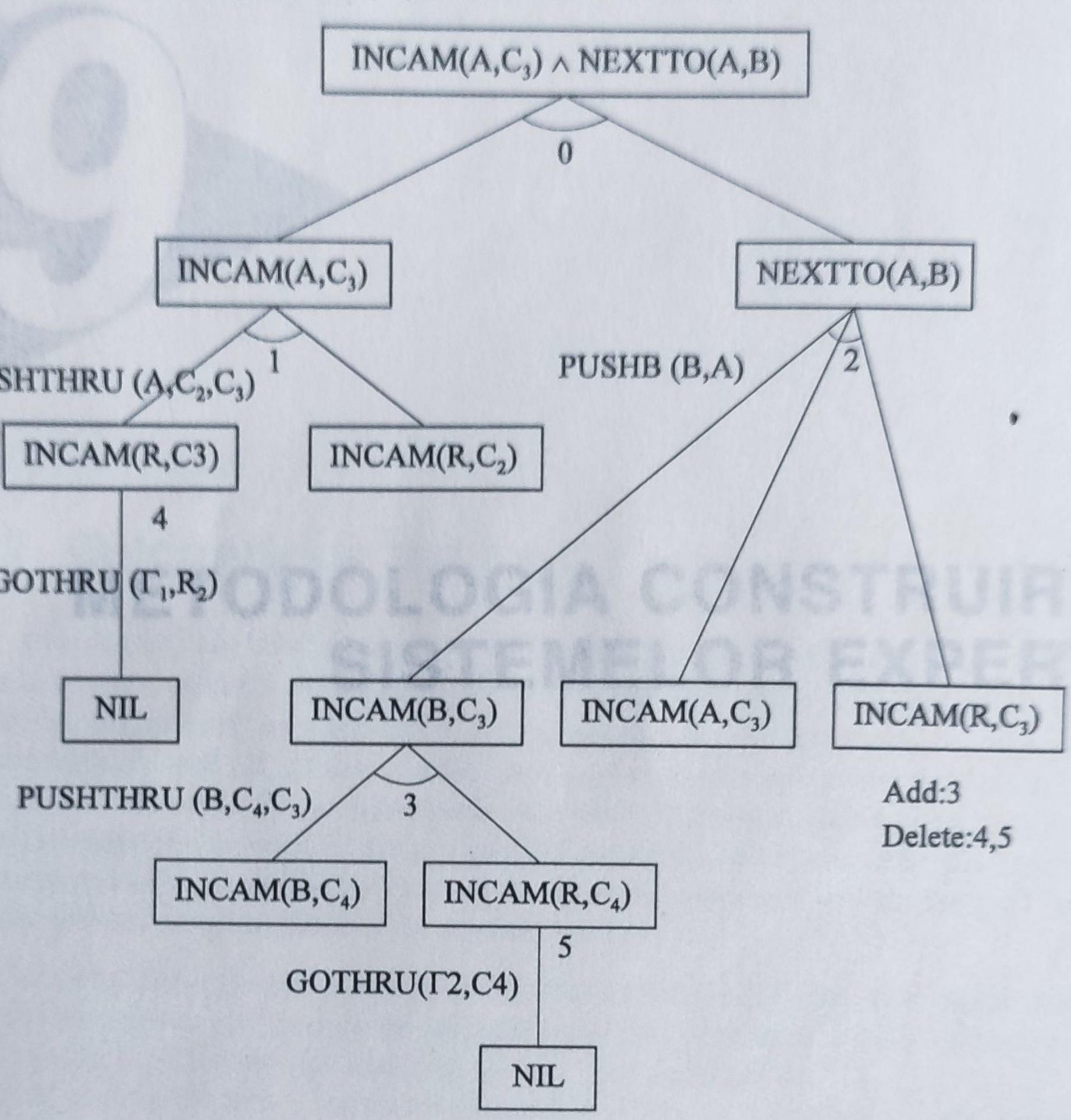
Nivel 3

Dacă se rafinează nivelul 4 prin adăugarea precondiţiilor nivelului 4 se obţine rezultatul din fig. 8.17. În această figură secvenţa operatorilor 1, 3, 2 la nivelul 4 este deja dată, aşa că timpul de căutare descreşte dacă se utilizează ca bază pentru construirea planului. Secvenţa de operatori a soluţiei este 4, 1, 5, 3, 2. Aceasta este:

GOTHRU(r_1, C_2) → PUSHTHRU(A, C_2, C_3)
→ GOTHRU(C_3, C_4) → PUSHTHRU(B, C_4, C_3)
→ PUSHB(B, A)

unde r_2 în fig. 8.17 este C_3 şi r_1 este oricare din camerele (fixat la C_1 în planul de nivel 1).

▼ ***Fig. 8.17. Arbore ŞI/SAU în spaţiul abstract de nivel 3***



Nivel 2

Cînd se compilează un plan mai detaliat în acelaşi mod, următorul plan este obţinut (cititotul poate verifica aceasta).

OPEN($U(r_1, C_2)$) → GOTHRU(r_1, C_2) → OPEN($U(C_2, C_3)$ →
PUSHTHRU(A, C_2, C_3) → OPEN($U(C_3, C_4)$) → GOTHRU(C_3, C_4) →
PUSHTHRU(B, C_4, C_3) → PUSHB(B, A)

Nivel 1

Rezultatul rafinării procesului este:

OPEN($U(r_1, C_2)$) → GOTHRU(r_1, C_2) → OPEN($U(C_1, C_2)$ →
GOTO($U(C_{2,C}3)$) → OPEN($U(C_2, C_3)$) → GOTOB(A) →
PUSHTHRU(A, C_2, C_3) → GOTOD($U(C_3, C_4)$) → OPEN($U(C_3, C_4)$) →
GOTHRU(C_3, C_4) → GOTOP(B) → PUSHTHRU(B, C_4, C_3) → PUSHB(B, A).

METODOLOGIA CONSTRUIRII SISTEMELOR EXPERT

9.1. Categorii de aplicaţii

Ne propunem să analizăm care sînt diferenţele dintre sisteme expert realizate pentru diferite aplicaţii, dacă acestea se pot grupa în clase, modul în care are loc transferul cunoştinţelor. La multe din aceste întrebări nu se poate da un răspuns net şi precis, însă, pot apărea elemente ce fac obiectul cercetărilor viitoare. Este necesară în acest context o analiză asupra limitărilor sistemelor expert, cît şi a posibilităţilor viitoare de evoluţie. Odată stabilite aceste chestiuni se va încerca să se definească principiile de bază ale metodologiei de construcţie a unui sistem expert.

Fără a propune o grupare a sistemelor expert foarte diferite în clase, putem face o diferenţiere, în funcţie de natura scopului urmărit, deci de utilizare. Astfel, aceste sisteme se pot împărţi în trei mari categorii:

a) SE de clasificare - interpretare, pot realiza spre exemplu clasificarea cauzelor posibile ale unei disfuncţionări (cantitative, calitative, vizuale) în scopul de a determina semnificaţiile acestor tipuri de date. Acest tip de SE este cel mai vechi (diagnostic, prospecţii) şi cel mai des utilizat. Dezvoltate cu precădere pentru aplicaţiile în medicina cercetările au influenţat aplicaţiile ulterioare rezultînd ceea ce numim diagnostic tehnic. Dintre sistemele utilizate în domeniul medical MYCIN este un exemplu perfect. Sistemul utilizat pentru diagnosticarea infecţiilor bacteriene utilizează o reprezentare a cunoştinţelor sub formă de reguli, descriind legăturile între simptome şi bolile posibile. Cunoştinţele folosite aici sînt cu precădere empirice şi de aceea mai sînt numite şi "cunoştinţe de suprafaţă". Este util să remarcăm faptul că sistemele ce utilizează cunoştinţe de suprafaţă nu pot să justifice raţionamentul ce explică deducţiile realizate. La nivelul raţionamentelor utilizate în MYCIN se disting două caracteristici principale:

- raţionare înainte (singurul mod).

- cercetare exhaustivă - aplicarea tuturor regulilor posibile pentru a creşte sau a diminua factorul de certitudine al concluziei.

Recent, cercetările s-au îndreptat spre alte sisteme capabile să utilizeze "cunoştinţe profunde". Dintre sistemele dezvoltate in domeniul diagnozei tehnice amintim DART Stanford University - 82, ACE, Bell - 83, MICRO EXPERT - ISIS, PDS - Carnegie University, PERITUS - Stanford Research Institute, SPEAR - DEC, RECONSIDER - University of California. În acest domeniu construirea unui sistem expert este foarte dificilă din cauză că regulile cu care operează trebuie să fie foarte generale astfel încât să fie asigurată portabilitatea de la o maşină la alta. Pentru diagnoza sistemelor de calcul cu aceiaşi structura remarcăm faptul că testele legate de memorie, magistrală rămîn identice. Cunoştinţele pentru astfel de aplicaţii sunt reprezentate prin descrieri structură-funcţie precum şi prin relaţiile dintre simptome şi defecte. Dacă pentru circuitele numerice descrierea relaţiilor intrare-ieşire oferea "cunoştinţe în profunzime" pentru circuitele complexe acest lucru nu mai este posibil. În această situaţie se întîlnesc atât "cunoştinţe de suprafaţă" cît şi "cunoştinţe profunde", însă sistemul expert nu are nevoie de "cunoştinţe profunde" complete şi foarte detaliate.

b) Sisteme expert de control (monotorizare) - sînt caracterizate prin introducerea noţiunii "timp" care este primordială, şi au sarcina de a supraveghea buna evoluţie a unui proces (conducere de procese). Aceste sisteme supraveghează evoluţia datelor sau semnalelor provenind de la procesul controlat. Ele tratează, deci date continue într-o logică nemonotonă ca în exemplul:

(dacă *A* este adevărat la un moment *T*, atunci *A* este lansat în execuţie la momentul $T + Q$).

Pentru aceste sisteme măsura intervalelor regulate de timp trebuie să fie interpretată, generînd declanşarea unei acţiuni imediate, atîta timp cât anumite condiţii sînt îndeplinite. Este necesară deci o structură de control ce permite evoluţia datelor altfel decît în cazul clasic al diagnosticării. Cunoştinţele pot fi reprezentate sub formă de tabele cu date care în funcţie de valoarea mărimii controlate selectează strategiile ce vor fi aplicate la momentele următoare. O categorie specială de control este cea destinată optimizării proceselor cooperative.

c) SE de anticipare. Aceste sisteme sînt caracterizate printr-o alocare a resurselor ţinînd cont de restricţii. Ex. CAO (anticiparea unui rezultat) pentru meteorologie, pentru anticiparea evoluţiei în viitor pe baza datelor prezente şi a evoluţiilor trecute, pentru ordonarea şi planificarea producţiei. Această compartimentare nu a fost făcută de dragul unei clasificări, ea permite studiul şi căutarea după caz a sistemului cadru pentru dezvoltarea unei aplicaţii. Spre exemplu, sistemele expert utilizate în diagnoză nu pot fi utilizate la dezvoltarea aplicaţiilor de conducere a proceselor.

În concluzie, este deci indispensabilă determinarea clasei aplicaţiei, şi căutarea sistemului de dezvoltare adecvat.

9.2. Alternative în construcţia unui sistem expert

Pentru construirea unui sistem sînt posibile două alternative:

- existenţa unei aplicaţii care în urma analizei justifică abordarea cu un sistem expert în detrimentul programării clasice;
- existenţa unui sistem expert cadru pentru o anumită categorie de aplicaţii la care se ataşează o bază de cunoştinţe specializată în problema ce se cere rezolvată.

Hotărîrea de a utiliza un sistem expert pentru rezolvarea problemei este luată după examinarea următoarelor situaţii:

- dacă o problemă este bine rezolvată prin modele numerice clasice (progrmarea liniară sau altele) este inutilă dezvoltarea unui sistem expert. Din contră, dacă problema nu este bine rezolvată în maniera clasică decît în condiţii ideale rar atinse, sau numărul parametrilor de natură simbolică puşi în joc este dificil de luat în consideraţie într-un model numeric, atunci se poate apela la o rezolvarea utilizînd sisteme expert. Unii autori consideră că sistemele expert sînt lipsite de rigoare în formalism, dar este bine de remarcat faptul că această construcţie nu este utilizată decît atunci cînd metodele clasice sînt ineficace. În general, ineficacitatea provine din faptul că problema este dificil formalizată din motive obiective sau subiective;
- existenţa expertului uman, recunoscut ca fiind persoana ce cunoaşte bine lucrul studiat. Aceasta explică de ce construcţia unui sistem expert depinde de un expert uman pentru baza sa de cunoştinţe şi de ce dacă problema nu este rezolvată de persoane nu poate fi rezolvată nici de un sistem expert;
- dacă evoluţia cunoştinţelor în domeniu este caracterizată de o dinamică rapidă, necesită frecvenţe schimbări, atunci un sistem expert se poate aplica. În această situaţie se poate folosi şi un program clasic, însă amendamentele aparent minore, aduse la acest program îi pot modifica profund comportamentul.

Realizarea programelor clasice este un proces rigid, greu, ce crează dificultăţi în elaborarea specificaţiilor, proces ce conduce la programe lipsite de supleţe şi fără posibilităţi de îmbunătăţire. Din contră, sistemele expert prin însăşi construcţia lor se pretează la modificări locale numeroase, în condiţiile în care modificările nu afectează coerenţa bazei de cunoştinţe:

- principalele calităţi ale expertului sînt exprimate în cunoştinţele transferate sistemului, cunoştinţe ce sînt în general de bun simţ, şi nu formalizate. Nu trebuie înţeles însă, că sistemul expert nu poate raţiona utilizînd cunoştinţe puternic formalizate, însă marele cîştig rezidă în utilizarea cunoştinţelor slab formalizate;

▼ ***Tabelul 9.1.***

Problema / Soluția	Precisă și stabilă	Precisă dar evoluează frecvent	Fluctuantă intr-un domeniu bine stabilit
Cunoscută	Programare clasică	SE uşor de actualizat ca urmare a evoluţiei	SE pentru că se poate adapta la fiecare problemă
Necunoscută	SE pentru găsirea soluţiei, apoi abandonat în favoarea programării clasice	SE pentru căutarea soluţiei, apoi adaptat pentru exploatare	SE pentru căutarea soluţiei, uşor de exploatat pentru că se adaptează la problema precisă

- problemele pentru a căror rezolvarea sînt necesare cîteva ore de muncă umană şi nu sînt repetitive, nu se pretează la rezolvarea cu sisteme expert pentru a căror rezolvare activitatea este mult mai complexă. Dacă, de exemplu, stabilirea unei configuraţii pentru un sistem de calcul necesită o zi de muncă umană, ţinînd cont de faptul că o problemă similară este rezolvată practic în fiecare zi, un sistem expert este adecvat. Un alt aspect într-o astfel de situaţie este marcat de imposibilitatea formulării corecte a cererii de către beneficiar, o mare parte din timp fiind ocupată cu discuţiile între specialist şi beneficiar. Utilizarea unei interfeţe operator "prietenoase" (limbaj natural) a facilitat rezolvarea unor astfel de programe, un exemplu fiind sistemul XCON utilizat la DEC;
- o cotă a rotaţiei de personal ridicată (cerută des în tehnică) este în aceiaşi măsură un semn că utilizarea sistemelor expert permite memorarea experienţei acumulate de persoanele ce lucrează în domeniu (perenitatea informării);
- cînd într-o clasă bine definită de probleme, nu se ştie care va fi rezolvarea precisă, variabilele de intrare, numărul de terminale, tipul lor, tipuri de comunicaţie numeroase nu permit scrierea programelor pentru toate variantele în forma clasică. Tabelul 9.1 sumarizează posibilele utilizări ale sistemului expert.

Ne propunem să răspundem la întrebarea: se poate folosi un mecanism de inferenţă existent, va trebui să-l scriem singuri? A doua cale este mai îmbietoare, autorul este în mijlocul problemei, poate cunoaşte toate detaliile programului şi este posibil de a adapta structura internă la necesităţile

proprii. Însă trebuie pus în evidenţă faptul că se realizează un sistem care într-o realizare profesională este necesar să dispună de toate facilităţile pentru dezvoltarea şi testarea bazei de cunoştinţe.

Pentru a încerca un răspuns la întrebarea: care este sistemul cadru cel mai adecvat, cercetătorii americani decid să facă un test a 8 mecanisme diferite (Emycin, KAS, Expert, OPS5, Rosie, RLL, Age şi Hearsay) analizînd avantajele şi inconvenientele lor. Pentru aceasta, ei cer echipelor specializate pe cîte unul din sisteme (echipele ce le-au dezvoltat) să dezvolte în 3 zile un mic sistem expert pentru detectarea, găsirea şi informarea persoanelor competente despre o eventuală scăpare într-un sistem de stocare a petrolului sau a altor produse chimice. S-au dedus astfel cîteva principii importante:

- sistemul trebuie să posede numai gradul de generalitate necesar pentru rezolvarea problemei date. Altfel spus, afirmaţia "cine poate mult poate şi puţin" este o eroare, un sistem prea general posedă capacităţi neutilizate, şi aceasta în detrimentul performanţelor;
- este necesar ca sistemul cadru să aibă o serie de caracteristici, cum sînt:
  - Limbajul de reprezentare a cunoştinţelor la nivelul utilizatorului trebuie să fie cît mai simplu şi universal;
  - Un mijloc de a ajunge la mecanismul de control dacă generalitatea este mai importantă decît eficacitatea;
  - Capacităţi de dialog elaborat (limbaj cvasinatural - dacă timpul de elaborare nu este critic);
- în măsura posibilului se vor utiliza sisteme cadru care au servit la aplicaţii similare.

Aceste criterii sînt mai mult empirice decît metodologii ale unei teorii precise, însă sînt impuse de faptul că cercetările în domeniu sînt încă incipiente. Pe de altă parte, puţine sînt sistemele cadru ce au fost utilizate la suficiente sisteme expert pentru a le testa limitele, eficacitatea şi adecvanţa. Aşa cum s-a încercat să se arate pe parcursul paragrafului prezent, proiectantul are de ales între două alternative:

a) utilizarea unui sistem cadru existent cu avantajul existenţei mecanismului de inferenţă şi a subsistemului cognitiv, dar cu dezavantajul rezultat din necunoaşterea intimităţii sistemului. Mai sus s-a încercat sistematizarea unor criterii pentru alegerea unui astfel de sistem cadru după domeniul aplicaţiei;
b) construirea completă a sistemului expert.

Este uşor de remarcat că un sistem cadru ce conferă o mare generalitate (mai ales în inferfaţa cu utilizatorul) asigură o realizare mai rapidă. Problema esenţială care depinde în speţă de calitatea alegerii rămîne totuşi performanţă, nu totdeauna satisfăcătoare. Construirea unui sistem dedicat asigură totuşi mari avantaje, marcate prin supleţe şi performanţe deosebite, însă necesită un buget de timp pentru elaborare mult mai ridicat. Sistemul astfel construit va putea fi complet validat numai după o lungă perioadă de

exploatare. Odată găsite mecanismele de construcţie, principala activitate constă în dezvoltarea bazei de cunoştinţe prin transferul experienţei de la om la maşină.

9.3. Etape ale realizării unui sistem expert

Paralel cu derularea procesului de transfer al experienţei procesul de realizare a unui sistem expert poate fi descompus în trei perioade:

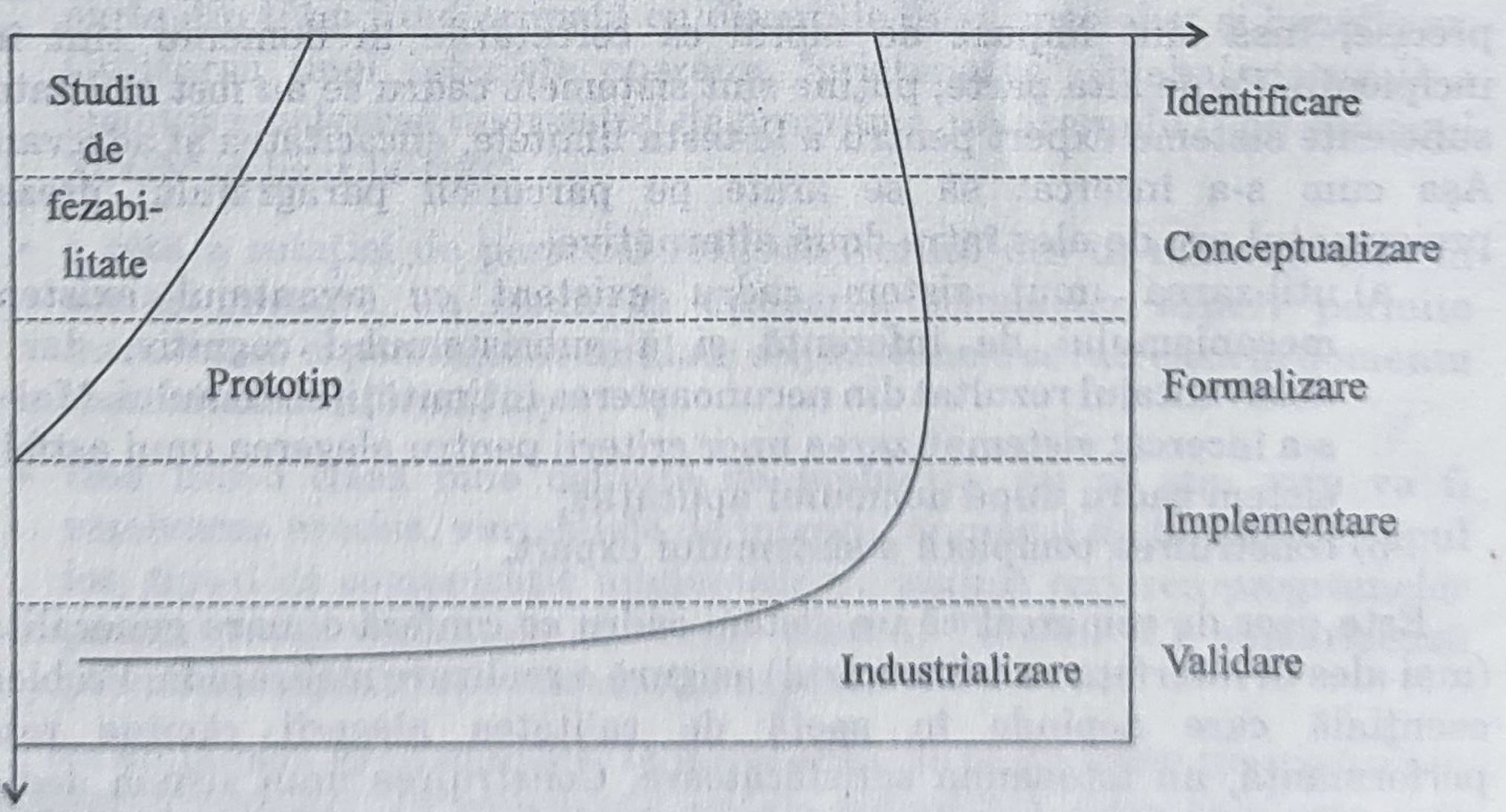
A) Un studiu de fezabilitate realizat mai întîi cu ajutorul unei machete de sistem expert;

B) Realizarea prototipului - vizează construirea unui prototip pe care se experimentează deferitele moduri de reprezentare a cunoştinţelor, mecanismele de inferenţă realizate;

C) În final, sistemul expert este realizat pornind de la prototip. Dacă rezultatele sînt satisfăcătoare şi structura sistemului realizat ca prototip, fie pornind de la zero, permite obţinerea eficacităţii maxime, urmează etapa de industrializare.

În fig. 9.2 se reprezintă cele cinci faze aferente transferului de cunoştinţe în conjuncţie cu cele trei etape de dezvoltare. Aceasta arată că studiul de fezabilitate este foarte restrîns, însă dezvoltarea prototipului şi experimentarea reprezintă părţi importante ale procesului de construcţie. Diferenţa între machetă şi prototip nu poate fi foarte clară, însă, precis prin prototip se înţeleg subansambluri ale sistemului expert final pe cînd macheta este total independentă.

▼ *Fig. 9.2. Etapele dezvoltării sistemului expert*



Etapa de prototip permite verificarea rapidă (două sau patru luni) dacă pentru problema dată se justifică utilizarea sistemului expert. Ea presupune realizarea unei machete care nu rezolvă decît cîteva cazuri tipice. Macheta este realizată fără a urmări aspecte legate de optimizarea timpului de răspuns şi a memoriei ocupate, sau de validarea formalismului utilizat. Inginerul de cunoştinţe utilizează pentru aceste reprezentări simple şi flexibile, nuclee existente fără să urmărească validarea formalismului utilizat.

La finalul acestui studiu, problema trebuie să fie clar definită, permiţînd să se ia o decizie asupra desfăşurării ulterioare a lucrării. Soluţii distincte ale problemei trebuiesc testate şi comparate pe parcursul dezvoltării prototipului (care poate fi o extensie a machetei). Pentru aceasta, cel mai comod de utilizat sînt sistemele expert cadru. Acest prototip trebuie, în majoritatea cazurilor să rămînă modest în principiu, el va fi abandonat atunci cînd inginerul de cunoştinţe are o idee precisă asupra dezvoltărilor viitoare ale sistemului expert pe baza experienţelor realizate. În final, este preferabilă abandonarea prototipului, modificările nu pot garanta eficacitatea sistemului, chiar dacă prototipul este la un moment dat mult mai bine cunoscut.

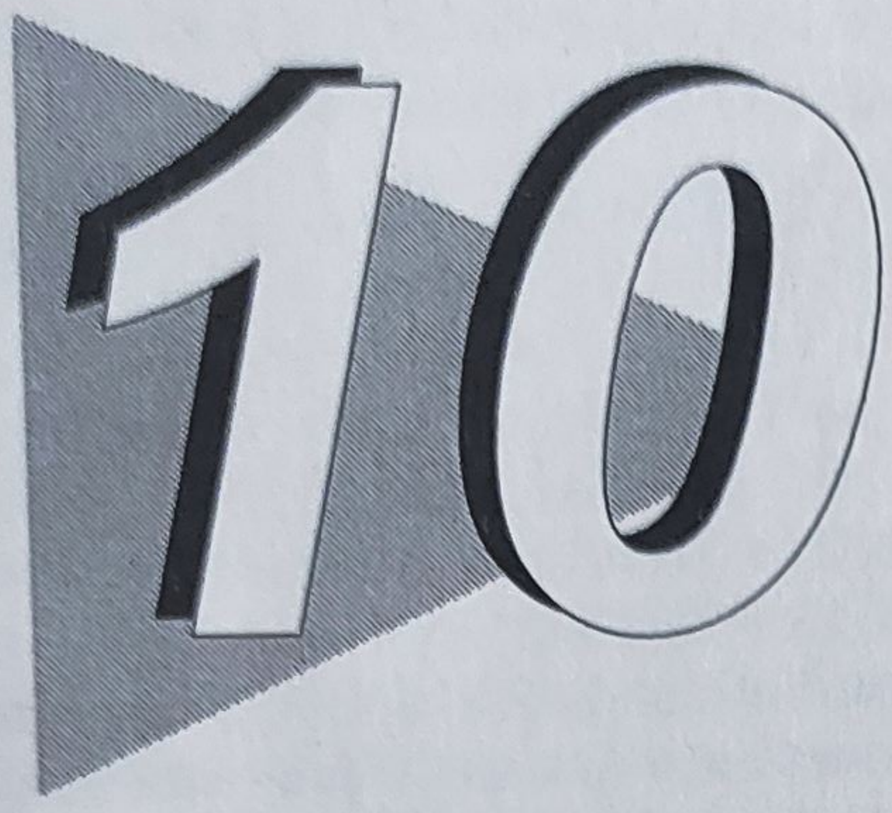
Pentru ilustrare să considerăm o comparaţie imaginară: extensia unui prototip (sau a unei machete) este similară cu aceea de a mări un cart pentru a obţine o maşină de formula I. Aceasta nu va atinge niciodată performanţele unei maşini construite în acest scop, deficienţele provin din însăşi arhitectura cartului. Diferenţa esenţială între această etapă şi etapele precedente rezidă în faptul că la acest nivel se ştie foarte exact ce facem şi cum facem. Se procedează acum la o rescriere a prototipului testat în etapa anterioară urmărind satisfacerea condiţiilor de eficacitate. Baza de cunoştinţe continuă să evolueze pe parcurul acestei faze încorporînd cazuri reale şi din ce în ce mai complexe (pentru aceasta este necesară dispunerea de un modul de explicare la acest nivel). Sistemul expert începe să devină utilizabil înainte ca baza de cunoştinţe şi obiectivul fixat iniţial să fie realizat complet.

Dezvoltarea în varianta clasică descompune problema în subprobleme realizate independent unele de altele. Din contră, abordarea de viitor este abordarea experimentală şi incrementală a machetei, prototipului, sau a tuturor elementelor. Se observă în varianta clasică descompunerea în subprobleme cu rezolvarea acestora în paralel. Astfel rezolvarea este limitată, subproblemele fiind parcurse independent. Construcţia printr-un proces experimental şi incremental este bazată pe luarea în consideraţie la anumite etape a tuturor subproblemelor într-o manieră detaliată.

Multe companii se găsesc astăzi la sfîrşitul etapei *A*, o machetă a fost realizată, însă problemele nu au fost clar delimitate. Dacă decizia de trecere la etapa *B* era luată (şi multe sînt deja luate) partea cea mai importantă începe aici. Ca efect, un studiu asupra domeniului va fi necesar şi indispensabil. El permite determinarea cunoştinţelor utilizate de expertul uman şi maniera de a le reprezenta, sprijinindu-se pe experienţa deja

acumulată în acest domeniu (faza de conceptualizare, formalizare). Această etapă este importantă căci ea condiţionează urmarea studiului. Lucrurile care sînt efectuate (reprezentarea cunoştinţelor, utilitare, mecanism de inferenţă) determină caracteristicile sistemelor expert viitoare şi performanţele lor. Primele rezultate ale studiului pe teren, permit enunţarea criteriilor de căutare a utilitarelor utilizate în dezvoltarea sistemului expert.

Prototipul obţinut, deci baza de cunoştinţe nu trebuie să fie voluminoasă, ea serveşte doar ca suport de decizie pentru autorizarea trecerii la faza următoare. Etapa *C* începe prin determinarea utiliarelor corespunzătoare realizării finale (ce pot fi aceleaşi cu cele ale fazei *B*). Realizarea acestei etape se sprijină pe prototipul construit în etapa precedentă insistînd pe eficacitatea performanţelor. Baza de cunoştinţe a SE final se măreşte lent pe măsura avansării proiectului, prin noi cunoştinţe şi prin rezultatele testelor ce demonstrează validitatea coerenţei cunoştinţelor. Trebuie amintit că faza de extracţie a cunoştinţelor este lungă şi delicată, ea consumă o mare parte din timpul total de realizare a sistemului expert.



APLICAŢII

10.1. Sistem expert pentru supravegherea evoluţiei mărimilor controlate

Supravegherea buclelor de reglare aferente conducerii oricărei instalaţii industriale este o problemă delicată cu atît mai dificilă cu cît complexitatea procesului este mai mare. Analizînd evoluţia mărimilor controlate se poate observa că în multe cazuri funcţionarea este apropiată regimurilor de prealarmare, alarmare, atingerii limitelor tehnologice. Evoluţia în astfel de zone determină costuri de producţie mari cu efecte nedorite. Este necesar ca în aceste situaţii operatorul să fie avizat şi să poată interveni în sensul corectării. Apare din aceste considerente ca necesară utilizarea unui sistem expert ce oferă avantaje în adaptabilitatea la situaţii concrete cît şi înglobarea cunoştinţelor tipice izvorîte ca urmare a experienţei de exploatare pentru fiecare caz în parte. De asemenea, cu cît complexitatea procesului creşte şi numărul parametrilor supravegheaţi este mai mare posibilităţile de urmărire a evoluţiei acestora prin simpla inspectare a istoriei este mai greoaie. Apare deci firească opţiunea de dezvoltare a unui pachet de programe expert pentru supravegherea evoluţiei.

Se poate considera că în aplicaţiile de conducerea proceselor de complexitate mică şi medie o structură modulară formată din pînă la 32 de controllere locale ce pot comunica cu un nivel ierarhic superior, nivel ce preia funcţiile specifice consolei operatorului de proces, este adecvată. Prin intermediul fiecăruia dintre controllere pot fi implementaţi de la simplii algoritmi P, PI, PD, PID, la algoritmi adaptivi şi optimali. Oricare ar fi soluţia aleasă la nivelul controllerelor locale SE trebuie să răspundă aceloraşi deziderate.

10.1.1. Structura sistemului

Comunicaţia între echipamentul destinat consolei operatorului de proces şi controllerele locale asigură actualizarea permanentă a bazei de date asociată fiecărei bucle. Se consideră că baza de date constituită la nivelul consolei operator are o structură modulară şi conţine toate informaţiile de interes asociate buclelor locale. Primii 256 octeti sînt asociaţi identificatorilor de instalaţie, bucle, mărimi. În continuare într-o structură identică pentru fiecare controller se constituie o bază de date, fiecărui controller fiindu-i alocate blocuri de 256 octeti. Deplasamentul în blocuri de 256 octeti în baza de date destinată stocării mărimilor de interes asociate fiecărei bucle este dată de adresa de identificare a controllerului pe linia serială. În situaţiile în care o structură de acest tip nu permite rezolvarea unei soluţii de automatizare complexă mai multe structuri modulare pot fi interconectate la nivelul consolei operator cu un nivel ierarhic superior.

Structura tipică a oricărui sistem expert presupune existenţa a două mecanisme esenţiale: baza de cunoştinţe (parte componentă a sistemului cognitiv) şi mecanismul de inferenţă (parte a sistemului rezolutiv). Din motive uşor de înţeles legate de execuţia on-line a părţii rezolutive, baza de cunoştinţe este creată off-line şi apoi transportată pe maşina ţintă în forma în care este utilizată de către mecanismul inferenţial.

10.1.2. Reprezentarea cunoştinţelor

Pentru crearea bazei de cunoştinţe a fost realizat un editor cu facilităţi grafice, editor ce crează structura de reguli sub forma de flat-file similar cu informaţiile stocate în fişiere de date utilizate în mod curent cu programele de baze de date.

Structura clausală implementată este de tip clausa Horn de forma:

$$A_1 \text{ and } A_2 \text{ and.,,,,.and } A_n \vdash B \text{ în care:}$$

- termenii $A_1, A_2, \ldots, A_n$ formează premisa regulii constituită în partea IF;
- termenul B formează concluzia constituind partea THEN.

Pentru aplicaţia prezentată un termen din premisa poate avea una din structurile:

nume_param_1 | op_mat | valoare (1)
nume_param_2 | op_mat | val_procentuală*nume_param_2 (2)

în funcţie de specificul operaţiei realizate.

Pentru a asigura o uniformitate a păstrării informaţiei ce formează baza de cunoştinţe informaţiile inroduse sînt păstrate înr-un fişier cu următoarea structură:

RULE (nr_reg, tip, nume_param_1, op_mat, conect_1, valoare, identif_operator, nume_param_2, conect_2)

în care:

nr_reg este de tip numeric cu trei digiţi, cîmp ce asigură memorarea numărului de regulă între 1 şi 999;

tip cîmp de tip caracter (1 octet) ce conţine fie caracterul I ce indică primul termen din premisă, fie caracterul *A* pentru a indica faptul că termenul din premisă este legat de cel anterior prin conectiva logică .and..

nume_param_1 de tip şir caractere ce conţine numele simbolic al primului parametru utilizat în operaţie;

op_mat cîmp de tip caracter ce conţine unul dintre caracterele <, <=, >, >=, = ce indică tipul operaţiei de comparaţie;

conect_1 de tip logic în care "true" indică operaţia de comparaţie imediată şi "false" indică o comparaţie după înmulţimea parametrului 2 cu un factor;

valoare reprezintă un număr cu valoarea 1 în cazul operaţiilor de comparaţie imediată sau între 1 şi 99, cînd se calculează procentual din al doilea parametru;

identificator_operator ce are asociat "blanc" în cazul comparaţiei imediate sau simbolul % pentru a indica valoarea procentuală;

nume_param_2 de tip şir caractere ce conţine numele simbolic al celui de al doilea parametru utilizat în operaţie;

conect_2 de tip logic cu valorile "true" dacă urmează o nouă condiţie în premisă sau "false" dacă este ultima condiţie din premisa regulii.

Structura uniformă prezentată mai sus, similară cu cea a fişierelor asociate unei baze de date asigură uşurinţa vizualizării şi modificării bazei de cunoştinţe utilizând un program de baze de date obişnuit. În timpul creării bazei de cunoştinţe sînt asigurate facilităţi de ştergere şi corecare a regulilor deja introduse. Concluziile regulilor sînt prezentate într-o formă similară cu cele ale termenilor premisei, inferenţierea poate duce la obţinerea de noi valori logice asociate termenilor premisei.

10.1.3. Implementarea sistemului pentru achiziţia cunoştinţelor

Implementarea sistemului cognitiv ce crează baza de cunoştinţe este realizată în *C++*. Sursa simplificată a programului de creare a bazei de cunoştinţe este prezentată mai jos.

```
# include <stdio.h>
# include <string.h>
# include <graphics.h>
# include <conio.h>
# include <ctype.h>
# include <math.h>
# include <stdlib.h>
# include <dir.h>

// definirea structurii datelor într-o condiţie din clauza IF

struct inreg{
   int flag;
int nrreg;
char tip;
```

```
   char p1[11];
   char opm;
   char legv;
   int val;
   char proc;
   char str[5];
   char p2[11];
   char legl;
};

/* O premisă în forma clausală este constituită din pînă la 6 termeni
cu structură similară, definită în structura inreg */
struct inreg*iff,*and1,*and2,*and3;
struct inreg*and4,*and5,*and6;

//declaraţii de funcţii utilizate

void up(int x, int y, char *s);
void down(int x, int y, char *s);
void cit_par(int x, int y,);
void cit(int x, int y, char ss[10]);
void ecran(void);
void regula(void);
void afişare(void);
void load_rec(void);
void del(void);
void incarc(struct inreg*m);
void mr(int y, struct inreg*m);
char c[2], pathd [MAXPATH], patht[MAXPATH];
char tx[40], s[4];
int caled (void);
int calet (void);
int nreg = 0, 1 = 0, tip; /*0: numeric, 1: param.*/
FILE *t,*dbf;

/* Proceduri ce determină existenţa fişierelor ce crează baza de
cunoştinţe */
int caled()
{
   int done;
   struct ffblk ffblk;
   strcpy(pathd, "C:\\");
   getcwd(pathd, MAXPATH);
   strcat(pathd, "\\exp.dbf");
   done=findfirst(pathd, &ffbflk, 0);
   return(done);
}
int calet()
{
   int done;
   struct, ffblk ffblk;
   strcpy(patht, "C:\\");
   getcwd(patht, MAXPATH);
   strcat(patht, "\\text.txt");
   done=findfirst(patht, &ffblk, 0);
   return(done);
}
/* Procedura ce realizează încărcarea unei condiţii din structura de
tip IF într-un fişier similar cu un fişier de baze de date */

void incarc(struc inreg*m)
{
   int fl, i = 0, j;
   int nr;
```

```
char con;
char pp1[11];
char op;
char 11v;
int v;
char pr;
char pp2[11];
char 11;
fl=(unsigned char) m -> flag;
fwrite(&fl, siseof(fl), 1, dbf);
fprintf(dbf, "%3d", m -> nrreg);
con = m -> tip;
fwrite(&con, sizeof(con), 1, dbf);
strcpy(pp1, m -> p1);
while(pp1p[i]! = '\0') i++;
for(j = i; j <= 9; j++)
  pp1[j] = ' ';
pp1[j] = '\0';
fwrite(&pp1, sizeof(pp1), 1, dbf);
op = m -> opm;
fwrite(&op, sizeof(op), 1, dbf);
11v = m -> legv;
fwrite(&11v, sizeof(11v), 1, dbf);
fprintf(dbf, "%5d", m -> val);
pr = m -> proc;
fwrite(&pr, sizeof(pr), 1, dbf);
strcpy(pp2, m -> p2);
i = 0;
while(pp2[i]! = '\0') i++;
  for(j = i; j <= 9; j++)
    pp2[j] = ' ';
pp2[j] = '\0';
fwrite(&pp2, sizeof(pp2), 1, dbf);
11 = m -> legl;
fwrite(&11, sizeof(11), 1, dbf);
}
/* Procedura de formare a regulii în forma clauzală şi stocarea
acesteia în scopul generării bazei de cunoştinţe */

void load_rec(void)
{
  int i = 0, j;
incarc(iff);
if (1==2) incarc(and1);
if (1==3)
{  incarc (and1);
  incarc (and2);
}
if (1==4)
{  incarc (and1);
  incarc (and2);
  incard (and3);
}
if (1==5)
{  incarc (and1);
  incarc (and2);
  incarc (and3);
  incarc (and4);
}
if (1==6)
{  incarc (and1);
  incarc (and2);
```

```
      incarc (and3);
      incarc (and4);
      incarc (and5);
    }
  if (1==7)
    {  incarc (and1);
      incarc (and2);
      incarc (and3);
      incarc (and4);
      incarc (and5);
      incarc (and6);
    }
   while (tx[i]! = '\0') i++;
     for(j = i; j <= 39; j++)
       tx[j] = ' ';
   tx[j] = '\0';
   fwrite(&tx, sizeof(tx), 1, t);
}
/* Procedura ce realizează interfaţa cu utilizatorul la crearea bazei
de cunoştinţe */

void ecran(void)
   {char ss[3];
   setcolo(11);
rectangle(268, 12, 370, 32);
setcolor(11);
outtextxy(275, 20, "RULE NO:");]
itoa(nreg, ss, 10);
outtextxy(344, 20, ss);
setfillstyle(1, 0);
bar(0, 179, 640, 480);
up(10, 180, "If");
up(10, 220, "Then");
up(10, 260, "And");
setfillstyle(1, 11);
bar(0, 300, 64, 314);
setcolor(4);
rectangle(0, 299, 65, 315);
setcolor(15);
rectangle(0, 298, 66, 316);
setcolor(1);
outtextxy(2, 304, "CHOICE:");
   c[0] = getch();
   c[1] = '\0';
   c[0] = toupper(c[0]);
   while ((c[0]! = 73) && (c[0]! = 65) && (c[0]! = 84)) /* I, A sau
T */
     {
     c[0] = getch();
     c[1] = '\0';
     c[0] = toupper(c[0]);
     }
   outtextxy(57, 304, c);
}
/* Proceduri grafice utilizate de interfaţa utilizator */

void up(int x, int y, char *s)
   {
   setfillstyle(1,11);
   bar(x + 1, y + 1, x + 39, y + 14);
   setcolor(4);
   rectangle(x, y, x + 40, y + 15);
   setcolor(15);
```

```
line(x + 1, y - 1, x + 41, y - 1);
line(x + 2, y - 2, x + 42, y - 2);
line(x + 3, y - 3, x + 43, y - 3);
line(x + 41, y - 1, x + 41, y + 14);
line(x + 42, y - 2, x + 42, y + 13);
line(x + 43, y - 3, x + 43, y + 12);
setcolor(1);
if (s[0]=='I')
  outtextxy(x + 14, y + 4, s);
if (s[0]=='A')
  outtextxy(x + 8, y + 4, s);
if((s[0]! = 'I') && (s[0]! = 'A')) outtextxy(x + 4, y + 4, s);
}

void down(int x, int y, char *s)
  {  setfillstyle(1, 0);
    bar(x, y - 3, x + 43, y + 14);
    setfillstyle(1, 11);
    bar(x + 1, y + 1, x + 39, y + 14);
    setcolor(4);
    rectangle(x, y, x + 40, y + 15);
    setcolor(15);
    rectangle(x - 1, y - 1, x + 41, y + 16);
    setcolor(1);
    if (s[0]=='I')
      outtextxy(x + 14, y + 4, s);
    if (s[0]=='A')
      outtextxy(x + 8, y + 4, s);
    if((s[0]! = 'I') && (s[0]! = 'A')) outtextxy(x + 4, y + 4, s);
  }

void cit_par(int x, int y)
{
  setfillstyle(1, 11);
  bar(x + 100, y + 15, x + 279, y + 85);
  setcolor(4);
  rectangle(x + 99, y + 14, x + 280, y + 85);
  setcolor(15);
  line(x + 102, y + 13, x + 281, y + 13);
  line(x + 281, y + 13, x + 281, y + 83);
  setcolor(1);
  outtextxy(x + 102, y + 18, "param. 1:");
  outtextxy(x + 102, y + 33, "operator:");
  outtextxy(x + 102, y + 48, "param. numeric? [y/n]:");
}

void cit(intx, int y, char ss[10])
{
  int i = 0;
  char c1;
  setfillstyle(1, 11);
  bar(x, y + 5, x + 12, y + 8);
  setcolor(1);
  c1 = getch();
  while(c1! = 13)
  {
    if(i < 1) i = 0;
    if(c1==8)
    {
  ss[--i] = '\0';
  bar(x, y, x + 30, y + 8);
  }
  else{
```

```
      ss[i] = c1;
      i++;
      ss[i] = '\0';
    }
    outtextxy(x, y, ss);
    c1 = getch();
}
void mr(int y, struct inreg*m)
{
   int pozx, pozy, k = 0, i = 0;
   char str[10], cc1[2], cc2[2];
   m -> flag = 0x20;
   m -> nrreg = nreg;
   if (c[0]==73) /*I*/
     {  m -> tip = 'i';
        cit_par(30, 180);
        pozx = 30;
        pozy = 180;
     }
   if (c[0]==84) /*T*/
     {  cit_par(30, 220);
        pozx = 30;
        pozy = 220;
     }
  if (c[0]==65) /*A*/
    {  m -> tip = 'a';
       cit_par(30, 250)
       pozx = 30;
       pozy = 250;
    }
  setcolor(1);
  line(pozx + 194, pozy + 23, pozx + 202, pozy + 23);
  delay(800);
  cit(pozx + 190, pozy + 18, m -> p1);
  setcolor(1);
  line(pozx + 194, pozy + 38, pozx + 202, pozy + 38);
  delay(800);
  cit(pozx + 190, pozy + 33, cc1);
  m -> opm = cc1[0];
  setcolor(1);
  line(pozx + 268, pozy + 54, pozx + 274, pozy + 54);
  delay(800);
  cc2[0] = getch();
  cc2[1] = '\0';
  while ((cc2[0]! = 'y') && (cc2[0]! = 'n'))
    {  cc2[0] = getch();
       cc2[1] = '\0';
    }
  setcolor(11);
  line(pozx + 268, pozy + 54, pozx + 274, pozy + 54);
  setcolor(1);
  outtextxy(pozx + 268, pozy + 48, cc2);
  if (cc2[0]=='y')
    {  outtextxy(pozx + 102, pozy + 63, "valoare :");
       tip = 0; /* param. numeric */
    }
  if (cc2[0]=='n')
    {  outtextxy(pozx + 102, pozy + 60, "valoare :");
       outtextxy(pozx + 102, pozy + 73, "param. 2:");
       tip = 1; /* param. nenumeric */
```

```
  }
if (tip==0)
  {  setcolor(1);
    line(pozx + 190, pozy + 68, pozx + 198, pozy + 68);
    delay(800);
    cit(pozx + 190, pozy + 63, m -> str);
    m -> val = atoi(m -> str);
    m -> p2[0] = '\0';
    m -> legv = 'f';
  }
if (tip==1)
  {  setcolor(1);
    line(pozx + 190, pozy + 64, pozx + 198, pozy + 64);
    delay(800);
    cit(pozx + 190, pozy + 59, m -> str);
    while (m -> str[i]! = '\0') i++;
    if(m -> str(i - 1]=='%')
    {  m -> proc = '%';
      m -> str[i - 1] = '\0';
    }
    else m -> proc = ' ';
    m -> val = atoi(m -> str);
    setcolor(1);
    line(pozx + 190, pozy + 78, pozx + 198, pozy + 78);
    delay(800);
    cit(pozx + 190, pozy + 73, m -> p2);
    m -> legv = 't';
  }
setcolor(11);
outtextxy(216, 340, "ESC: return to the main menu");
cc2[0] = getch();
cc2[1] = '\0';
while (cc2[0]0! = 27)
  {  cc2[0] = getch();
    cc2[1] = '\0';
  }
setcolor(11);
rectangle(180, y - 3, 450, y + 12);
setfillstyle(1, 0);
bar(181, y - 3, 449, y + 12);
setcolor(11);
outtextxy(182, y, s);
outtextxy(213, y, m -> p1);
outtextxy(297, y, cc1);
outtextxy(315, y, m -> str);
while(m -> str[k]! = '\0') k++;
if (tip==1)
  {  if(m -> proc=='%') outtextxy(315 + k*8, y, "%");
    else outtextxy(315 + k*8, y, "*");
    outtextexy(315 + (k + 1)*8, y, m -> p2);
  }
m -> legl = 'f';
}
/* Procedura ce utilizează interfaţa grafică cu utilizatorul pentru
introducerea regulilor ce formează baza de cunoştinţe */

void regula(void)
{
  char cc[2], str[10], ch;
  int y, lung, i = 0;
  nreg++;
  ecra();
```

```
do
{
  switch(c[0])
  {
  case 'I':  down(10, 180, "If");
        strcpy(s, "IF");
        iff = (struct inreg*) malloc(sizeof(struct inreg));
        mr(52, iff);
        setcolor(11);
        line(180, 46, 450, 46);
        ecran();
        y = 50;
        break;
case 'A':  down(10, 260, "And");
        stricpy(s, "AND");
        if (1==1)
        {  iff -> legl = 't';
          and1 = (struct inreg*) malloc(sizeof(struct inreg));
          y = 65;
          mr(65, and1);
          ecran();
        }
        if (1==2)
        {  and1 -> legl = 't';
          and2 = (struct inreg*) malloc(sizeof(struct inreg));
          mr(80, and2);
          y = 80;
          ecran();
      }
      if (1==3)
      {  and2 -> legl = 't';
        and3 = (struct inreg*) malloc(sizeof(struct inreg));
        mr(95, and3);
        y = 95;
        ecran();
      }
      if (1==4)
      {  and3 -> legl = 't';
        and4 = (struct inreg*) malloc(sizeof(struct inreg));
        mr(110, and4);
        y = 110;
        ecran();
        }
        if (1==5)
        {  and4 -> legl = 't';
          and5 = (struct inreg*) malloc(sizeof(struct inreg));
          mr(125, and5);
          y = 125;
          ecran();
        }
        if (1==6)
        {  and5 -> legl = 't';
          and6 = (struct inreg*) malloc(sizeof(struct inreg));
          mr(125, and6);
          y = 140;
          ecran();
        }
        break;
}
  1++;
} while (c[0]! = 84); /*'T'*/
setcolor(11);
```

```
line(180, y + 14, y + 14);
strcpy(s, "THEN");
down(10, 220, "Then");
setfillstyle(1, 11);
bar(80, 222, 400, 232);
setcolor(4);
rectangle(79, 221, 401, 233);
setcolor(15);
rectangle(78, 220, 402, 234);
ch = getch();
while(ch! = 13)
{
   if(i < 1) i = 0;
   if(ch==8)
   {
     tx[--i] = '\0';
     bar(80 + i*8,222, 80 + (i + 1)*8,230);
   }
   else{
     tx[i] = ch;
     i++;
     tx[i] = '\0';
   }
   setcolor(1);
   outtextxy(80, 224, tx);
   ch = getch();
}
setfillstyle(1, 0);
bar(0, y + 15, 640, 480);
lung = 0;
while(tx[lung]! = '\0') lung++;
setcolor(11);
rectangle((588-lung*8)/2, y + 25, (666 + lung*8)/2, y + 40);
outtextxy((594-lung*8)/2, y + 29, "Then");
outtextxy((664-lung*8)/2, y + 29, tx);
setcolor(11);
outtextxy(240, 300, "All is correct? [Y/N]");
c[0] = getch();
c[1] = '\0';
c[0] = toupper(c[0]);
while((c[0]! = 'Y') && (c[0]! = 'N'))
   {  c[0] = getch();
     c[1] = '\0';
     c[0] = toupper(c[0]);
   }
if (c[0]==78) /*'N'*/
{  setfillstyle(1, 0);
   bar(200, 300, 420, 325);
   setcolor(11);
   outtextexy(228, 300, "Delete or Modify the rule?");
   cc[0] = getch();
   cc[1] = '\0';
   while ((cc[0]! = 100) && (cc[0]! = 109) /* d sau m */
   {  cc[0] = getch();
     cc[1] = '\0';
   }
   switch(cc[0])
   {  case 'd':  nreg--;
        1 = 0;
        setfillstyle(1, 0);
   bar(0, 0, 640, 480);
   break;
```

```
    case 'm':  nreg--;
               setfillstyle(1, 0);
               bar(200, 300, 440, 320);
               setcolor(11);
               outtextxy(208, 300, "To modify must retype the rule!");
               outtextxy(208, 315, "Please star with 'IF'!");
               getch();
               i = 0;
               regula();
               break;
     }
   }
   if (c[0]==89) /*'Y'*/
   {
     setfillstyle(1, 0);
     bar(0, 0, 640, 480);
   }
{
/* Procedura ce permite inspectarea bazei de cunoştinţe create */

void afişare(void)
{
   int i, nr, poz = 0, x = 1, y = 1, lg;
   char buf[40], s1[10], s[3];
   setfillstyle(1, 0);
   bar(0, 0, getmaxx(), getmaxy());
   printf("\n\n  If you want to delete some rules, please");
   printf("\n     memorate their number! OK?");
   getch();
   nr = 1;
   fseek(dbf, 321, SEEK_SET);
   fseek(r, 0, SEEK_SET);
   setfillstyle(1, 0);
   bar(0, 0, getmaxx(), getmaxy());
   setcolor(11);
   outtextxy(0, 5, itoa(nr, s1, 10));
   lg = strlen(s1);
   outtextxy(lg*8,5, ".");
   s1[0] = '\0';
   do {
     while((poz! = 32) || (s1[0]! = 'f'))
     {  fread(buf, 34, 1, dbf);
       s1[0] = buf[4];
       s1[1] = '\0';
       if(s1[0]=='i')
         {  gotoxy(x + 4, y);
           printf("If");
         }
       if(s1[0]=='a')
         {  x = 1;
           y++;
           gotoxy(x + 4, y);
           printf("And");
         }
       for(i = 5; i <= 15; i++)
         s1[i - 5] = buf[i];
       s1[i - 5] = '\0';
       gotoxy(x + 8, y);
       printf("%s", s1);
       s1[0] = buf[15];
       s1[1] = '\0';
       gotoxy(x + 18, y);
```

```
printf("%s", s1);
for(i = 17; i <= 21; i++)
  s1[i -- 17] = buf[i];
s1[i - 17] = '\0';
gotoxy(x + 20, y);
printf("%s", s1);
if (buf[16]=='t')
{  gotoxy(x + 26, y);
  if (buf[22]=='%') printf("%");
  else printf("*");
  for(i = 23; i <= 32; i++)
    s1[i - 23] = buf[i];
  s1[i - 23] = '\0';
  gotoxy(x + 27, y);
  printf("%s", s1);
 }
 poz = 32:
 s1[0] = buf[33];
 s1[1] = '\0';
}
poz = 0;
x = 5;
y++;
nr++;
fread(buf, sizeof(buf), 1, t);
buf[40] = '\0';
gotoxy(x, y);
printf("Then %s", buf);
if (y > 23)
{  printf("Press any key to continue...");
 getch();
 setfillstyle(1, 0);
 bar(0, 0, getmaxx(), getmaxy());
 gotoxy(1, 1);
}
nreg--;
x = 1;
y++;
setcolor(11);
if(nreg! = 0)
{  outtextxy(0, (y - 1)*13, itoa(nr, s2, 10));
  lg = strlen(s2);
  outtextxy(lg*8, (y - 1)*13, ". ");
 }  while (nreg! = 0);
 getch();
}
/* Implementarea mecanismului de ştergere a unor reguli existente */

void del(void)
{  int n, d[20], i, pozd = 0;
   char buf[33], ss[4];
   int fl, luncg, j = 0;
   char ch;
   setfillstyle(1, 0);
   bar(0, 0, getmaxx(), getmaxy());
   gotoxy(20, 2);
   printf("Delete any rule? [y/n]:");
   scanf("%c", &ch);
   while((ch! = 'y') && (ch! = 'n')) scanf("%c", &ch);
   if(ch=='y')
   {  fl = (char) 0x2A;
```

```
printf("\n Enter numers of rules, in ascendent order.");
printf("\n ESC = finish, BACKSPACE = delete. OK?");
ch = getch();
i = 0;
while(ch! = 27)
{
  do
  {
    ss[j] = getch();
    j++;
    if (ss[j - 1]==8) j--;
    } while(ss[j - 1]! = 13);
    ss[j] = '\0';
    printf("\n\r%s", ss);
    n = atoi(ss);
    d[i] = n;
    i++;
    ch = getch();
  }
i = 0;
fseek(dbf, 321, SEEK_SET);
lung = strlen(d);
while(lung! = 0)
{
  fread(buf, 34, 1, dbf);
  for(n = 0, n <= 2; n++)
    ss[n] = buf[n + 1];
  ss[n] = '\0';
  n = atoi(ss);
  if(n==d[i]);
  {
    fseek(dbf, pozd, SEEK_CUR);
    fwrite(&fl, sizeof(fl), 1, dbf);
  }
  i++;
  pozd+ = 34;
  lung--;
  }
 }
}
/* Programul principal pentru generarea regulilor de producţie */

void main(void)
{  char ch;
  int nri = 0;
  int gdr = DETECT, gm;
  initgraph(&gdr, &gm, "C:\\TC\\BGI");
  if (caled())
    {
    printf("\n File not found!");
    getch();
    exit(0);
    }
  if((dbf = fopen(pathd, "r + w")) = NULL)
    {  printf("\nCannot open file for reading/writing.");
      getch();
      exit(0);
    }
  if (calet())
  {
printf("\n File not found!");
getch();
```

```
    exit(0);
    }
  if ((t = fopen(patht, "r + w"))==NULL)
    {  printf("\nCan't read from file!");
    getch();
    exit(0);
    }
  fseek(dbf, 321, SEEK_SET);
  fseek(t, 0, SEEK_SET);
  ch = 121;
  while(ch! = 110) /*'N'*/
    {
    regula();
    load_rec();
    setcolor(11);
    outtextxy(240, 300, "Add a new rule? [Y/N]");
    ch = getch();
    while((ch! = 'y') && (ch! = 'n')) ch = getch();
    nri+ = 1;
    1 = 0;
    }
  fseek(dbf, 4, SEEK_SET);
  fwrite(&nri, sizeof(nri), 1, dbf);
  afişare();
  del();
  setfillstyle(1, 0);
  bar(220, 305, 400, 315);
  gotoxy(35, 19);
  printf("CIAO!");
  ch = getch();
  fclose(t);
  fclose(dbf);
  return 0;
  closegraph();
}
```

Se dau mai jos cîteva exemple de reguli de producţie utilizate:

```
Regula i:  IF: val_intrare > lim_sup_tr
           THEN: Traductor defect
Regula j:  IF: mod_lucru = AUTO
           and: Kp < 1
           and: Ti < 1
           and: EROARE > 90% ER_MAX
           and: TIMP >= 2Ti
           THEN: Evoluţie incorectă.
```

10.1.4. Implementarea mecanismului de inferenţă

Structura clausală Horn asigură minimizarea timpului de execuţie asociat mecanismului de inferenţă. Pentru premisa oricărei reguli, premisa ce conţine mai mulţi termeni legaţi prin conectiva logică .and. orice termen a cărei valoare inferenţiată este "false" determină asocierea valorii "false" pentru toată premisa, fapt ce duce la următoarea regulă şi continuarea procesului inferenţial. Rezultatul inferenţelor este prezentat operatorului sub forma unui

mesaj în urma căruia operatorul decide dacă rezultatul este efectiv aplicat sau nu. Programul sursă al mecanismului de inferenţa este prezentat în cele ce urmează:

```
/* Au fost omise liniile destinate headerelor şi declaraţiilor.
Operaţiile se execută pe baza datelor din baza de date asociată
controllerului în lucru */
char pathd[MAXPATH], patht[MAXPATH];
FILE*, dbf, *t;

void main(void)
   {  char buf[40], s1[10];
     char log, masca;
     int pozd = 0, pozt = 0, flag, i = 0;
     int val, val1, val2, j, nreg = 0;
     int gdr = DETECT, gm;
     initgraph(&gdr, &gm, "C: \\TC\\BGI");
     if (caled())
      {
      printf("\n File not found!");
      getch();
      exit(0);
      }
if((dbf = fopen(pathd, "r"))==NULL)
  {  printf("\nCannot open file for reading/writing.");
     getch();
     exit(0);
  }
if (calet())
  {
    printf("\n File not found!");
    getch();
    exit(0);
  }
if ((t = fopen(patht, "r"))==NULL)
  {  printf("\nCan't read from file!");
     getch();
     exit(0);
  }
setfillstyle(1, 0);
bar(0, 0, getmaxx(), getmaxy());
fseek(dbf, 289, SEEK_SET);
fseek(t, 0, SEEK_SET);
while(!eof(dbf))
  {  fread(buf, pozd + 34, 1, dbf);
     s1[0] = buf[0];
     s1[1] = '\0';
     flag = atoi(s1);
     if (flag! = 0x2A)
       {  for(i = 1; i <= 3; i++)
            s1[i - 1] = buf[i];
          s1[i - 1] = '\0';
          nreg = atoi(s1);
          for(i = 5; i <= 14, i++)
            s1[i - 5] = buf[i];
          s1[i - 5] = '\0';
          /* caută în baza de date;*/
          /* întoarce valoarea val1.*/
          /* (coresp. lui p1) */
        for(i = 17; i <= 21; i++)
          s1[i - 17] = buf[i];
```

```
        s1[i - 17] = '\0';
        val = atoi(s1);
        if (buf[16]=='t') /* urmează param. p2 */
        {   for(i = 23; i <= 32; i++)
            s1[i - 23] = buf[i];
          s1[i - 23] = '\0';
          /* preia val2 din bd(coresp. lui p2); */
          if (buf[22]=='%') val* = 0.01;
          val* = val2;
        }
        switch(buf[15])
          {
            case '<':  if(val1<val) log = 't'; /* compară val cu
                       val1.*/
                  else log = 'f';          /* întoarce log.*/
                  break;
            case '>':  if(val1>val) log = 't';
                  else log = 'f';
                  break;
            case '=':  if(val1==val) log = 't';
                  else log = 'f';
                  break;
          }
        if (log=='f')
          {
            while(val==nreg)
            {
              fread(buf, pozd + 34, 1, dbf);
              for(i = 1; i <= 3; i++)
                s1[i - 1] = buf[i];
              s1[i - 1] = '\0';
              val = atoi(s1);
              pozd+ = 34;
            }
          pozd-=34;
          pozt+=40;
        }
      if ((log='t') && (buf[33]=='f'))
        {  fread(buf, 40, 1, t);
          i = 0;
          while(buf[i]! = '\0') i++;
          setfillstyle(1, 11);
          bar((640 - (i - 1)*10)/2, 100, (640 + (i - 1)*10)/2, 130);
          setcolor(4);
          rectangle((640 - (i - 1)*10)/2 - 1, 99,
          (640 + (i - 1)*10)/2 + 1, 131);
          outtextxy(280, 110, "WARNING!");
          setcolor(15);
          rectangle((640 - (i - 1)*10)/2 - 2, 98,
          (640) + (i - 1),*10)/2 + 2, 132);
          setcolor(1);
          outtextxy((640 - (i - 1)*10)/2, 120, buf);
          exit (1);
        }
          }
          pozd+=34
        }
      }
```

10.1.5. Integrarea în aplicaţie

Întrucît procesul inferenţial este singurul ce se execută "online", el este organizat sub forma unui task cu prioritatea cea mai mică faţă de taskurile ce deservesc comunicaţia între consola operator şi controllerele conectate serial. Motivul pentru care acest task este privit ca un task de prioritate mică rezidă în faptul că activitatea critică rămîne cea de culegerea datelor. Taskul ce constituie mecanismul de inferenţă poate furniza rezultate la intervale de timp de aproximativ 10 ori mai mari faţă de perioada de eşantionare. Pentru ca modulul de inferenţă să fie instalat ca task este necesară activarea sa ca funcţie task într-o manieră similară cu cea descrisă mai jos:

```
void far task_mec_inf(void)
  {
  iniţilizări variabile locale
  while (condiţie_task)
    {
    task mec_inf
    }
  }
```

Rezultatele experimentale obţinute în mediu simulat (cu baza de date din proces simulată sub forma de fişiere) arată o comportare satisfăcătoare atît din punctul de vedere al timpului de execuţie, al comutării corespunzătoare a taskurilor cît şi a rezultatelor obţinute în urma inferenţelor.

10.2. Sistem expert pentru acordarea optimă a buclelor de reglare

Controllerele PID şi subseturile acestora P, PI, PD sînt foarte des utilizate în multe aplicaţii industriale. Găsirea parametrilor de stare constă în mod normal din trei paşi:

- Determinarea aproximativă a parametrilor de acord pornind de la una din formulele standard;
- Determinarea tipului răspunsului dorit şi selectarea unui criteriu ca "quarter-decay" or minimum integrated absolute error (IAE)";
- Realizarea ajustării necesare a parametrilor pentru obţinerea răspunsului dorit;

Operatorul poate evalua performanţele propriei setări prin modificarea referinţei şi observarea răspunsului simulînd în acest fel acţiunea perturbaţiilor.

Forma canonică a compensatoarelor din această categorie este descrisă prin relaţia:

$$G_1 = K_p\left(1 + sT_d + \frac{1}{sT_i}\right) \tag{10.1}$$

în care:

K_p - factor de proporţionalitate;
T_i - constanta timpului de integrare;
T_d - constanta timpului de derivare;

De cele mai multe ori stabilirea valorilor acestor parametrii, astfel încît performanţele să fie cele dorite este mai mult o artă decît o ştiinţă. Gradul de cunoaştere a părţii fixate influenţează puternic valorile parametrilor de acord. Una din tehnicile de acord bazată pe limita de stabilitate a sistemului este tehnica Ziegler-Nichols. Acordarea empirică după metoda Ziegler-Nichols porneşte de la eliminarea efectelor termenului integral şi derivativ după care se urmăreşte creşterea factorului de amplificare pînă cînd se atinge punctul critic de oscilaţie.

Mărimile factorului de amplificare (K_u) şi perioadei de oscilaţie (P_u) pentru limita de stabilitate vor fi utilizate la calculul parametrilor de acord după relaţiile empirice:

$$D_p = 0.6*K_u \quad T_i = P_u/2 \quad T_d = P_u/8 \tag{10.2}$$

Răspunsul la intrare treapta cu aceste valori de setare obţinute rapid, duce la valori ale suprareglajului neacceptabil de mari. O altă tehnică, cea analitică aplicată pentru sisteme caracterizate prin funcţia de transfer:

$$G_2(s) = \frac{Ke^{-T_n}}{(1 + sT_1)(1 + sT_2)} \tag{10.3}$$

duce la următorii parametrii de acord

$$K_p = 2(T_i + T_2)/3KT,\ T_i = T_1 + T_2,\ T_d = T_1T_2/(T_1 + T_2) \tag{10.4}$$

De asemenea, în situaţia în care procesul controlat este greu de modelat, cum este cazul multor sisteme industriale acordarea este bazată de cele mai multe ori pe intuiţie, fără o justificare complet teoretică. În plus, este normal ca operatorii să aibă propriile preferinţe pentru cea mai bună traiectorie a răspunsului, fapt ce va determina modificarea parametrilor de acord în sensul obţinerii răspunsului dorit. În acest caz nu se mai pune problema definirii matematice a optimalităţii. Este cunoscut faptul, că mulţi dintre operatori consideră ca principale performanţe perioada de oscilaţie şi suprareglajul.

Descrieri ale diferitelor sisteme expert ce realizează ajustarea parametrilor în controllere PID au apărut în literatură. Una din cele mai cunoscute lucrări T.W. Kraus, T.J. Myron - 1984 tratează modul de ajustare prin care se urmăreşte obţinerea celui mai rapid răspuns cu respectarea valorilor maxime permise pentru suprareglaj şi dumping. Alte sisteme urmăresc determinarea erorii integrale şi calculul factorului de amplificare după identificarea sistemului, alegerea diferiţilor algoritmi numerici de control bazaţi pe obiectivele operatorului sau utilizarea regulilor de clasificare în scopul determinării clasei din care face parte instalaţia. O excelentă lucrare J. Litt -

1990 încearcă să surprindă dependenţa între performanţe şi valorile parametrilor de acord. Abordarea problemei de acordare optimă impune două mari direcţii: fundamentarea unei metode de ajustare a parametrilor şi integrarea sistemului expert în pachetul de programe cu funcţionare în timp real.

10.2.1. Dependenţe între parametrii şi performanţe

În vederea utilizării unui sistem expert pentru ajustarea parametrilor o serie de rezultate ale procesului de simulare au fost necesare. În acest sens s-a studiat răspunsul unei bucle de reglare pentru care partea fixată este descrisă prin funcţia de transfer:

$$G_2(s) = \frac{e^{-T_r}}{(T_1 s + 1)(T_2 s + 1)} \qquad (10.5)$$

considerînd că acoperă o gamă largă de instalaţii industriale. În urma studiului răspunsului prin simulare, cît şi prin metode teoretice bazate pe caracteristicile de frecvenţă pot fi obţinute dependenţele între performanţe şi parametrii de acord. Pentru a obţine dependenţele între valorile parametrilor T_v, T_d, K_p şi forma traiectoriei răspunsurile au fost desenate în planul T_d/P_u-P_u/T_i (fig. 10.1). Normalizarea axelor prin P_u ajută la dimensionare şi este generală pentru o clasă de sisteme de tipul 10.5. Aşa că dacă o instalaţie poate fi aproximată printr-o funcţie de transfer similară cu 10.5 şi cu perioada de oscilaţie determinată experimental, parametrii de acord doriţi pentru un anumit tip de răspuns pot fi selectaţi direct din diagramele corespunzătoare. La implementarea prin utilizarea unui sistem expert aceste diagrame sînt utilizate calitativ şi nu cantitativ în sensul extragerii dependenţelor între performanţe şi parametrii de acord. Fiecare set de caracteristici este determinat pentru valori diferite ale factorului de amplificare K_p.

Aprecierea adecvenţei răspunsului este caracterizată de o serie de mărimi cu semnificaţia cunoscută în teoria sistemelor.

1. *Suprareglaj* σ_1.
2. Raportul între valoarea corespunzătoare celui de al doilea şi primul maxim numit şi *decrement* $\delta = \sigma_3/\sigma_1$.
3. *Timpul de creştere* t_c.
4. *Dumping* $\psi = (\sigma_3 - \sigma_2)/(\sigma_1 - \sigma_2)$ acest factor este mare dacă răspunsul este puternic oscilant şi mic dacă răspunsul este caracterizat de oscilaţii mici.
5. *Perioada oscilaţiilor* (P_0).
6. *Timpul regimului tranzitoriu* ce nu este specificat direct, mărimea sa fiind automat cuprinsă în valorile mărimilor δ şi ψ.

Din analiza răspunsului pentru diferiţi parametrii de acord pot fi deduse o serie de reguli privind dependenţa între mărimile de calitate formulate mai sus şi parametrii de acord.

▼ *Fig. 10.1. Caracterizare răspunsuri*

Observaţiile calitative sînt:

1. Creşterea lui K_p creşte suprareglajul şi invers.
2. Creşterea lui K_p scade perioada oscilaţiilor şi invers.
3. Creşterea lui K_p creşte dumpingul şi invers.
4. Creşterea lui K_p scade timpul de creştere şi invers.
5. Creşterea lui K_p înrăutăţeşte stabilitatea.
6. Creşterea lui K_p creşte decrementul şi invers.
7. Scăderea lui T_i creşte suprareglajul şi invers.
8. Scăderea lui T_i creşte dumping şi invers.
9. Scăderea lui T_i înrăutăţeşte stabilitatea.
10. Scăderea lui T_i creşte timpul regimului tranzitoriu.
11. Creşterea lui T_d scade timpul de creştere şi invers.
12. Creşteea lui T_d îmbunătăţeşte stabilitatea şi invers.

10.2.2. Strategia de ajustare a parametrilor

Modificarea parametrilor de acord în conformitate cu performanţele răspunsului trebuie să determine apropierea de răspunsul dorit. În acest sens pentru sistemul expert este necesară precizarea valorilor numerice ale mărimilor de calitate dorite (suprareglaj, dumping etc.), ponderea dată de utilizator pentru importanţa mărimii de calitate, valorile iniţiale ale parametrilor de acord. Strategia de modificare este bazată pe calculul erorii normalizate prin diferenţa între valoarea curentă (obţinută din răspunsul sistemului) şi valoarea dorită, împărţită la valoarea dorită. În acest mod eroarea este scalată indiferent de unităţile de măsură. Pe baza valorii erorii normalizate şi a semnului său se vor determina parametrii ce trebuiesc modificaţi şi în ce sens. Întrucât de cele mai multe ori se obţine un conflict între direcţiile de ajustare a parametrilor după performanţele dorite, se va alege acea direcţie de modificare ce determină impactul maxim. În acest scop eroarea normalizată va fi multiplicată prin ponderea de importanţă a criteriului şi cu +1 dacă impactul dorit se determină prin creşterea parametrului de acord sau cu –1 dacă acesta se obţine prin micşorarea parametrului de acord.

Notînd:

E_i - valoarea erorii normalizate a mărimii i

W_i - ponderea asociată mărimii de calitate i

S_i - semnul asociat (+1 sau –1) după influenţa parametrului de acord asupra performanţelor se obţine:

$$m_i = S_i * W_i * E_i \qquad (10.6)$$

pentru fiecare mărime de calitate i. Acest termen este puternic dependent de valoarea ponderii W_i. Este normal ca în ponderile asociate, suprareglajul să aibă pondere maximă.

O altă problemă este legată de stabilirea valorii incrementului sau decrementului ce determină noua valoare a parametrului de acord. O strategie

posibilă, utilizată în tehnicile de optimizare, conduce la folosirea unei secvenţe Fibonacci $a_k = a_{k-1} + a_{k-2}$ pentru fixarea pasului sau la metode de bisectare cu şirul $a_k = 2^k$. Se obţine pentru noua valoare a parametrului de acord:

$$P(k+1) = P(k)\left[1 + \frac{1}{a_k}\operatorname{sgn}\left(\sum_{i=1}^{R} m_i\right)\right] \qquad (10.7)$$

în care:

$P(k+1)$ - noua valoare a unuia din parametrii: K_p, T_i, T_d;

$P(k)$ - valoarea veche;

a_k - termenul şirului de ordinul k;

Σm_i - suma scorului pentru fiecare mărime de calitate calculată conform (10.6);

sgn(x) - semnul lui x (–1 dacă $x < 0$, 0 dacă $x = 0$, 1 dacă $x > 0$);

Termenul $s = \Sigma s_i W_i E_i$ reprezintă şi scorul care este o măsură a apropierii de optim. Se poate considera că dacă s este mai mic decît o valoare impusă (dată iniţial de utilizator) comportarea în bucla închisă este corespunzătoare şi nu este necesară intervenţia asupra parametrilor. Dacă scorul rămîne aproximativ constant la un număr de iteraţii cu o valoare mai mare decât cea impusă indică atingerea unui extrem local. Pentru ca procesul iterativ să continue se va scădea K_p şi creşte T_i, (parametrii ce influenţează puternic suprareglajul) odată cu resetarea ordinului şirului pentru fixarea pasului de modificare.

10.2.3. Integrarea unui sistem expert cu aplicaţii în timp real

Programele ce operează în timp real au caracteristici ce pot fi rezumate astfel:

- necesitatea de reactualizare simultană a parametrilor (în paralel), a parametrilor fizici specifici sistemului, cît şi a resurselor;
- necesitatea de răspuns în restricţiile de timp impuse de proprietăţile fizice ale sistemului real la care calculatorul este cuplat.

În general, pot fi listate cîteva probleme cînd un sistem expert este utilizat în timp real:

- necesitatea unui interval mare de timp pentru obţinerea unei soluţii prin inferenţă. Din acest moiv o cerinţă deosebită în realizarea SE cu aplicaţii în timp real impune noi moduri de reprezentare a cunoştinţelor în sensul scăderii timpului de rulare;
- inferenţa este întreruptă de evenimente externe asincrone ce necesită prelucrarea unor date;
- pentru tratarea a doua sau mai multe evenimente cu aceeaşi prioritate, SE trebuie să fie capabil să execute inferenţe în paralel;

- multe din datele noi, pot fi contradictorii cu cele deja alocate. Din acest motiv este necesar ca baza de cunoştinţe să nu aibă o construcţie monolitică;
- valorile fizice continue provenite de la senzori vor fi transformate în simboluri logice cu care se operează.

Ideea de bază a integrării unui SE într-o aplicaţie în timp real porneşte de la organizarea întregii activităţi sub controlul unui sistem de operare în timp real în care SE este privit ca un task. În acest mod SE devine o parte a pachetului complet de programe în timp real. Evenimentele reactualizate sînt de diferite tipuri şi diferă numai după locul în care apar. Din acest punct de vedere se împart în:

- ♦ evenimente externe (exterioare SE) - de regulă datele provenite de la procesul fizic ce sînt obţinute în urma tratării întreruperilor;
- ♦ evenimente interne (interioare SE) ce sînt semnale cunoscute de la sistemele de operare însă nu sînt manipulate în paralel dar sînt necesare în execuţia taskului asociat sistemului expert. Dintre acestea se menţionează:
  - valorile din baza de cunoştinţe;
  - starea curentă a procesului inferenţial;
  - un semnal de ceas intern.

Toate aceste evenimente sînt asincrone, două sau mai multe dintre ele pot apare în acelaşi moment. Sistemul expert trebuie să fie capabil să proceseze datele de intrare cu prioritatea adecvată. Prin tratarea sistemului expert ca un task la sistemele în timp real se vor utiliza facilităţile sistemului de operare în tratarea evenimentelor menţionate. Manipularea întreruperilor şi activarea taskurilor în funcţie de evenimentele apărute este realizată de sistemul de operare.

10.2.4. Implementarea

Sistemul propus în această lucrare este construit din mai multe controllere locale capabile să implementeze algoritmi de conducere PID, conectate prin legătura serială la un calculator compatibil IMB-PC ce îndeplineşte şi funcţia de consola operator de proces. Parametrii de acord pot fi stabiliţi fie de la panoul frontal al fiecărui controller, fie de la consola operator care este şi mediul de implementare a sistemului expert. Prin legătura serială la nivelul consolei se constituie o bază de date asociată tuturor buclelor de reglare. În această bază de date se găsesc toate informaţiile de interes necesare procedurii de acordare optimă. Pentru fiecare controller baza de date asociată are aceeaşi structură uşurînd astfel accesul. În vederea analizei răspunsului se va genera un buffer circular ce conţine valoarea mărimii de ieşire la ultimile N momente de eşantionare. Numărul eşantioanelor este dependent de perioada oscilaţiilor procesului. Aşa cum s-a prezentat anterior decizia impune

explorarea răspunsului obţinut la fiecare set de parametri. Baza de cunoştinţe este creată cu ajutorul unui editor de reguli specializat ce crează fişierul text (cunoştinţele necesare) şi fişierul de reguli (baza de cunoştinţe). Reprezentarea prin reguli de producţie a fost aleasă atît datorită clarităţii cît şi a uşurinţei de implementare. Se dau mai jos exemple de reguli de producţie:

```
Rule i.        If parametrul este Kp
                 (and) Eroarea suprareglaj > 0
                 [sgn Wσ Eσ = 1]
               Then sgn Wσ Eσ = -1
Rule i + 1.    If parametrul este Kp
                 (and) Eroare perioada oscilaţii < 0
               Then sgn (WpoEpo = -1)
```

Pentru fiecare parametru de acord sînt create reguli specifice pe baza datelor obţinute din diagramele prezentate în fig. 10.1. Pe baza acestor reguli este creată baza de cunoştinţe cu informaţii ce au structura

(parametru de acord, mărime de calitate, semn eroare, sens modificare parametru)

Astfel pentru regulile ilustrate mai sus se obţine:

(K_p, sigma, 1, –1)

(K_p, P_0, –1, –1) în care,

+1 înseamnă semn pozitiv şi –1 semn negativ.

Se dă mai jos sursa C pentru achiziţia cunoştinţelor referitoare la regulile de producţie, a ponderilor asociate criteriilor de performanţă şi a valorilor impuse.

```
# include <stdio.h>
# include <conio.h>
# include <stdio.h>
# include <string.h>

# define END 79
# define UP 72
# define HOME 71
# define DOWN 80
# define ENTER 13
# define INSERT 82

void fereastra (int left, int top, int right, int bottom);

void main(void)
{
   int sigma, /* suprareglajul */
       delta, /* raportul suprareglajelor */
       xi, /* dumping-ul/
       po, /* perioada oscilaţiilor */
       tc, /* timpul de creştere */
   int w_sigma, /* pondere suprareglaj */
       w_delta, /* pondere raport supra reglaje */
       w_xi, /* pondere dumping */
       w_po, /* pondere perioada de oscilaţie */
       w_tc, /* pondere timp de creştere */

   int i, j, j_olf, len;
   int left=1, top=1, right=50, bottom=8;
   int left1, top1, right1, bottom1;
```

```
int last, lastx_if, lastx_them last_insert=0;

FILE *parametri, *tabele;
char ch;
char *ptr_begin, *ptr_end;
char *key_word[]={   "sgn",
                     "(",
                     ")",
                     "E_sigma",
                     "w_sigma * E_sigma",
                     "E_delta",
                     "w_delta" * E_delta",
                     "E_po",
                     "w_po * E_po",
                     "E_tc",
                     "w_tc * E_tc",
                     "E_xi",
                     "w_xi * E_xi",
                     " = +1",
          };
   char *title_section[]={   "REGULILE PT. Kc,
                             "Ti",
                             "Td",
          }
/* char *param[]={   "sigma"
                     "delta"
                     "po",
                     "tc",
                     "xi"
          }                  */
/*   char   temp_str1[20], temp_str2[20], temp_str3[20], string1[100],
          string2[100];
   temp_str1[0] = 0;
   temp_str2[0] = 0;
   temp_str3[0] = 0;
   string1[0] = 0;
   string2[0] = 0;
   clrscr();
/* INTRODUCEREA PARAMETRILOR IMPUŞI */
   textcolor(WHITE);
   left = 1;
   top = 1;
   right = 50;
   bottom = 8;
   fereastra(left, top, right, bottom);
   window(left, top, right, bottom);
   textbackground(RED);
   textcolor(WHITE);

   gotoxy(2,   2);   cprintf   ("Introduceţi   valorile   dorite   ale
parametrilor:");
   textbackground(LICHTBLUE);

   gotoxy(2, 3);
   cprintf("Suprareglaj: "); cscanf ("%d", &sigma);
   gotoxy(20, 3); cprintf("Pondere suprareglaj:");
   csacanf("%d", &w_sigma);

   gotoxy(2, 4); cprintf( "Delta :");
   cscanf("%d", &delta);
   gotoxy(20, 4); cprintf ("Pondere delta :");
   csacanf("%d", &w_delta);
   gotoxy(2, 5); printf( "Perioada :"); cscanf("%d", &po);
```

```
gotoxy(20, 5); cprintf("Pondere perioada :");
cscanf("%d", &w_po);

gotoxy(2, 6); cprintf( "Timp creştere:"); cscanf("%d", &tc);
gotoxy(20, 6); cprintf("Pondere t. creştere:");
cscanf("%d", &w_tc);

gotoxy(2, 7); cprintf( "Dumping :"); cscanf("%d", &xi);
gotoxy(20, 7); cprintf("Pondere dumping :");
csacanf("%d", &w_xi);
   parametri = fopen ("PARAM. TXT", "wt");
   fprint(parametri, "%6d%6d\n%6d%6d\n%6d%6d\n%6d%6d\n%6d%6d\n",
       sigma, w_sigma, delta, w_delta, po, w_po, tc, w_tc, xi, w_xi);
   tabele=fopen("TABEL.TXT", "wt");
/* CUVINTE DEDICATE */
   textbackground(BLACK); textcolor(WHITE);
   window(1, 1, 80, 25);
   left=52; top=1; right=78; bottom=23;
   fereastra(left, top, right, bottom);

   window(left, top, right, bottom);
   textbackgroud(RED); textcolor(WHITE);
   gotoxy(2, 2); cprintf("CUVINTE DEDICATE");
   textbackground(LICHTBLUE);
   for(i = 0; i < 15; i++)
   {
    gotoxy(2, i + 3);
    cprintf("%s", key_word[i]);
   }
   gotoxy(2, 3); textbackgroud(WHITE); textcolor(BLACK);
   cprintf("%s", key_word[0]);

   window(1, 25, 80, 25); gotoxy (1, 1);
   textbackground(WHITE); textcolor(RED);
   cprintf("HOME"); gotoxy(5, 1); textcolor(BLACK);
   cprintf(" Termina regula"); gotoxy(21, 1);
   textcolor(RED); cprintf("END"); textcolor(BLACK);
   gotoxy(24, 1);
   cprintf("Termina grup reguli"); gotoxy(45, 1);
   textcolor(RED); cprintf("INSERT"); gotoxy(51, 1);
   textcolor(BLACK);
   cprintf("Terminare IF"); gotoxy(67, 1); textcolor(RED);
   putch(0x18);
   textcolor(BLACK); gotoxy(68, 1); cprintf("Up");
   gotoxy(72, 1);
   textcolor(RED);
   putch(0x19);
   gotoxy(73, 1);
   textcolor(BLACK);
   cprintf("Down");

   textbackground(LICHTBLUE); textcolor(WHITE);
   window(1, 1, 80, 25);
   left1=1; top1=10; right1=50; bottom1=23;
   fereastra(left1, top1, right1, bottom1);
   getch();
   for(i = 0; i < 3; i++)
   {
    window(left1 + 1, top1 + 1, right1 - 1, bottom1 - 1);
    clrscr();
    gotoxy(2, 2); cprintf("%s", title_section[i]);
    fprintf(parametri, "%s\n", title_section[i]);
    window(1, 1, 80, 25);
```

```
j = 1; j_old = 0;
last = 0;
while(1)
  {
  if(j > j_old)
    {
    window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
    gotoxy(2, 2); cprintf("Regula%d", j);
    fprintf(parametri, "Regula%d\n", j);
    gotoxy(2, 3); cprintf("IF ");
    cprintf(parametri, "IF "); lastx_if=18;
    gotoxy(2, 5); cprintf("THEN");
    lastx_then=8; gotoxy(8, 3);
    j_old+j;
    }
chr=getch();
if(ch==0)
  {
  ch=getch();
  if(ch==END)
    {
    left=52; top=1; right=78; bottom=23;
    window(left, top, right, bottom);
    textbackgroud(LIGHTBLUE); textcolor(WHITE);
    gotoxy(2, last+3); cprintf("%s", key_word[last]);
    last=0;
    textbackground(WHITE); textcolor(BLACK);
    gotoxy(2, last+3); cprintf("%s", key_word[last]);
    textbackground(LICHTBLUE); textcolor(WHITE);
    last_insert=0;
    fprintf(parametri, "\n\n");
    break;
    }
switch(ch)
  {
  case UP:
    left=52; top=1; right=78; bottom=23;
    window(left, top, right, bottom);
    _setcursortype(_NOCURSOR);
    gotoxy(2, last+3);
    textbackground(LICHTBLUE);
    textcolor(WHITE);
    cprintf("%s", key_word[last]);
    is(last==0) last=14;
    else last--;
    gotoxy(2, last+3);
    textbackground(WHITE); textcolor(BLACK);
    cprintf("%s", key_word[last]);
    textbackground(LICHTBLUE);
    textcolor(WHITE);
    break;
  case DOWN:
    left=52; top=1; right=78; bottom=23;
    window(left, top, right, bottom);
    _setcursortype(_NOCURSOR);
    gotoxy(2, last+3);
    textbackground(LICHTBLUE);
    textcolor(WHITE);
    cprintf("%s", key_word[last]);
    is(last==14) last=0;
    else last++;
    gotoxy(2, last+3);
```

```
        textbackground(WHITE);
        textcolor(BLACK);
        cprintf("%s", key_word[last]);
        textbackground(LICHTBLUE);
        textcolor(WHITE);
        break;
      case HOME:
        gotoxy(8, 7);
        cprintf("Totul este corect[y/n]:");
        ch=getch();
        if(ch=='y')
          {
          j_old=j;
          j++;
          left=52; top='; right=78; bottom=23;
          window(left, top, right, bottom);
          textbackground(LICHTBLUE);
          textcolor(WHITE);
          gotoxy(2, last+3)
          cprintf("%s", key_word[last]);
          last=0;
          textbackground(WHITE);
          textcolor(BLACK);
          gotoxy(2, last+3);
          cprintf("%s", key_word[last]);
          textbackground(LICHTBLUE);
          textcolor(WHITE);
          window(left1 + 8, top1 + 3, right1 - 2, bottom1 - 2);
          clrscr();
          last_insert=0;
          strcat(string1, "\n");
          strcat(string2, "\n");
          fprintf(parametri, "%s%s", string1, string2);
          ptr_begin=strchr(string1, '_');
          ptr_end=strchr(string1, ')');
          len=(ptr_end - ptr_begin) - 1;
          strncpy(temp_str1, ptr_begin+1, len);
          temp_str1[len]=0;
          ptr_begin=strchr(string1, '=');
          strncpy(temp_str2, ptr, begin+1, 3);
          temp_str2[3]=0;
          ptr_begin=strchr(string2, '=');
          strncpy(temp_str3, ptr_begin+1, 3);
          temp_str3[3]=0;
          fprintf(tabele, "%4s%8s%4s%4s\n", sect[i], temp_str1,
          temp_str2, temp_str3);
          string[0]=0; string2[0]=0;
          break;
          }
        else
          {
          window(left1 + 8, top1 + 3, right1 - 2, bottom1 - 2);
          clrscr();
          last_insert=0;
          window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
          gotoxy(2, 2); cprintf("Regula%d", j);
          gotoxy(8, 3);
          lastx_if=8; lastx_then=8;
          string1[0]=0; string2[0]=0;
          break;
          }
      case INSERT:
```

```
            if(last_insert==0)
               {
               last_insert++;
               gotoxy(8, 5);
               strcat(string2, "THEN");
               }
            else
               last_insert=0;
            break;
            default: cprintf("ALTELE"); break;
            }
         }
         if(ch==ENTER)
           {
           if(last_insert==0)
             {
             window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
             _setcursortype(_NORMALCURSOR);
             gotoxy(lastx_if, 3);
             cprintf("%s", key_word[last]);
             strcat(string1, key_word[last]);
             lastx_if+=strlen(kei_word[last]);
             gotoxy(lastx_if, 3);
             }
           if(last_insert==1)
                {
                window(left1 + 2, top1 + 2, right1 - 2, bottom1 - 2);
                _setcursortype(_NORMALCURSOR);
                gotoxy(lastx_then, 5); cprintf("%s", key_word[last]);
                strcat(string2, key_word[last]);
                lastx_then+=strlen(key_word[last]);
                gotoxy(lastx_then, 5);
                }
             }
           }
         window(1, 1, 80, 25);
         }
         fclose(parametri);
         fclose(tabele);
         while(!kbhit());
}
void fereastra(int left, int top, int right, int bottom)
{
  int i;
    gotoxy(left, top);
    putch(0xc9);
    for(i = left + 11 i < right; i++)
    {
      gotoxy(i, top);
      putch(0xcd);
    }
    putch(0xbb);

    for(i = top + 1; i < bottom; i++)
    {
      gotoxy(left, i); putch(0xba);
      gotoxy(right, i); putch(0xba);
    }
         gotoxy(left, bottom); putch(0xc8);
         for(i = left + 1; i < right; i++) putch(0xcd);
         gotoxy(right, bottom); putch(0xbc);
}
```

Mecanismul de inferenţă după inspectarea bazei de date determină caracteristicile răspunsului şi calculează scorul. Dacă scorul este mai mic decât cel impus în baza de cunoştinţe nu se intervine asupra parametrilor de acord. Scorul este calculat ca $S_i W_i E_i$ şi indică apropierea faţă de răspunsul dorit. Valorile noilor parametrii de acord sînt obţinute pe baza relaţiei (10.7) şi setate la nivelul controllerului local. Fiecare iteraţie creşte ordinul şirului a_k ce intervine în relaţia (10.7). Sursa pentru mecanismul inferenţial ce se integrează ca un task este:

```
/* Declaraţii şi definiri de variabile */

void    determin_extrem (int far *point, int *er1, int *er2, int
        *er3, int *pozm1, int *pozm2, int *pozm3);
void round_ max(int far *point, int *ermax, int *pozmax);
void  round_next_max(int far *point, int poz_last_max, int *ermax,
      int *pozmax);
void round_mint(int far*point, int poz_last_max, int *ermin, int
*pozmin);
void modif_suma(int, i, float er, int w, float *suma);

void main(void)
{
   unsingned int far *point_round;
   unsingned int far *point_round;
   unsingned char far *point;
   unsingned char far *point1;

   char nr_bucla;
   float suma;
   int Kc, Ti, Td;
   int Kc_new Ti_new, Td_new, sgn;
   int sigma1, sigma2, sigma3;
   float po_inst, tc_inst, delta_inst, xi_inst;
   int pozmax1, pozmax2, pozmax3, pozmin;

   int sigma_ref;
   float po_ref, tc_ref, delta_ref, xi_ref;
   int w_sigma, w_delta, w_po, w_tc, w_xi;
   float er_sigma;
   float er_delta, er_po, er_tc, er_xi;

   int fibo[] = {2, 3, 5, 8, 13, 21};
   int i, j;
   char car;
   FILE *fp1, *fp2;
   char    total_string[100],    Kc_string[30],    Ti_string[30],
Td_string[30];
   char ch;
   clrscr();

/* Comunicaţia cu bufferul ce conţine istoria răspunsului nu este
dată întrucât este particulară modului de stocare */

   fp1 = fopen("PARAM.TXT", "rt");
   fp2 = fopen("TABEL.TXT", "rt");
   fseek(fp2, OL, SEEK_SET);
   i = 0; j = 0;
while((ch = fgetc(fp2)) != 'z')
{
  buf[i][j] = ch;
  j++;
  if(j%21 == 0)
```

```
    {
      j = 0;
      i++;
    }
  }
  determin_extrem  (point_round,  &sigma1,  &sigma2,  &sigma3,
                    &pozmax1, &pozmin, &pozmax2);
  fseek(fp1, OL, SEEK_SET);
  fscanf(fp1, "%6d%6d", &sigma_ref, &w_sigma);
  fseek(fp1, OL, SEEK_SET);
  fseek(fp1, 13L, SEEK_SET);
  fscanf(fp1, "%f%d%c%f%d%c%f%d%c%f%d", &delta_ref, &w_delta, &car,
  &po_ref, &w_po, &car, &tc_ref, &w_tc, &car, &xi_ref, &w_xi);
  if(sigma1 > sigma_ref)
  {
    er_sigma = (float)(sigma1 - sigma_ref);
    if(sigma2 >= 0)
    {
      sigma2 = 0;
      er_sigma = er_po = er_tc = er_xi = 0;
    }
    else
    {
    po_inst = ((pozmax1 - pozmin) *2* T_Esant)/1000.0;
    tc_inst = (po_inst * PERCENT_Po)/100.0;
    er_po = po_inst-po_ref;
    er_tc = tc_inst-tc_ref;
    if(sigma3 <= 0)
    er_delta = er_xi = 0;
    else
    {
      delta_inst = ((float)sigma3)/sigma1;
      xi_inst = ((float)(sigma3 - sigma2))/(sigma1 - sigma2);
      er_delta = delta_inst - delta_ref;
      er_xi = xi_inst - xi_ref;
    }
  }
  i = 0;
  suma = 0;
  while(strncmp(buf[i]," Kc", 4) == 0)
    {
    if(strncmp(buf[i] + 7, "sigma", 5) == 0)
     {
     modif_suma(i, er_sigma, w_sigma, &suma);
     i++;
     }
    if(strncmp(buf[i] + 7, "delta", 5) == 0)
     {
     modif_suma(i, er_delta, w_delta, &suma);
     i++;
     }
     modif_suma(i, er_delta, w_delta, &suma);
     i++;
     }
    if(strncmp(buf[i] + 7, " po", 5) == 0)
     {
     modif_suma(i, er_po, w_po, &suma);
     i++;
     }
    if(strncmp)buf[i] + 7, " tc", 5) == 0)
     {
     modif_suma(i, er_tc, w_tc, &suma);
```

```
    i++;
    }
  if(strncmp(buf[i] + 7, " xi, 5) == 0)
    {
    modif_suma(i, er_xi, w_xi, &suma);
    i++;
    }
  }
if(suma<0) sgn = -1;
else sgn = 1;
Kc=*((int*)(point+POINTER_Kc));
Kc_new = (int (Kc * (1 + 0.5* sgn));
itoa(Kc_new, Kc_string, 10);
suma = 0;
while( strncmp(buf[i], "  Ti", 4) == 0)
  {
    if(strncmp(buf[i] + 7, "sigma", 5) == 0)
    {
    modif_suma(i, er_sigma, w_sigma, &suma);
    i++;
    }
    if(strncmp(buf[i] + 7, "delta", 5) == 0)
    {
    modif_suma(i, er_delta, w_delta, &suma);
    i++;
    }
    if(strncmp(buf[i] + 7, "  po", 5) == 0)
    {modif_suma(i, er_po, w_po, &suma);
    i++;
    }
    if(strncmp(buf[i] + 7, "  tc", 5) == 0)
    {
    modif_suma(i, er_tc, w_tc, &suma);
    i++;
    }
    if(strncmp(buf[i] + 7, "  xi", 5) == 0)
    {modif_suma(i, er_xi, w_xi, &suma);
    i++;
    }
  }
if(suma<0) sgn = -1;
else sgn = 1;
Ti= *((int*)(point+POINTER_Ti));
Ti_nes = (int)(Ti* (1 + 0.5* sgn));
itoa(Ti_new, Ti_string, 10);
suma = 0;
while( strncmp(buf[i], "  Td", 4) == 0)
{
  if(strncmp(buf[i] + 7, "sigma", 5) == 0)
  {
    modif_suma(i, er_sigma, w_sigma, &suma);
    i++;
  }
  if(strncmp(buf[i] + 7, "  po", 5) == 0)
  {
    modif_suma(i, er_po, w_po, &suma);
    i++;
  }
if(strncmp(buf[i] + 7, "  tc", 5) == 0)
{
  modif_suma(i, er_tc, w_tc, &suma);
  i++;
```

```
    }
    if(strncmp(buf[i] + 7, "  xi", 5) == 0)
    {
      modif_suma(i, er_xi, w_xi, &suma);
      i++;
    }
  }

  if(suma<0) sgn = -1;
    else sgn = 1;
    Td = *((int*)(point+POINTER_Td));
    Td_new = (int)(Td * (1 + 0.5* sgn));
    itoa(Td_new, Td_string, 10);
    sprintf(total_string,    "NEWKc=%6s    NEWTi=%6s    NEWTd=%6s",
    Kc_string, Ti_string, Td_string);
    strcpy((poin+POINTER_Buf), total_string);
    printf("REZULTATELE EXPERT: \n %s\n", total_string);
  }
  else
    strcpy((point+POINTER_Buf),   "Nu   sînt   necesare   modificări
    parametri");
}

void modif_suma(int i, float er, int w, float *suma)
{
    if(er>0)
{
  if(buf[i][14] == '+')
  {
    if(buf[i][18] == '-')
      *suma = *suma - w*fabs(er);
    else
      *suma = *suma + w*fabs(er);
  }
}
else
{
  if(buf[i][14] == '-')
  {
    if(buf[i][18] == '-')
      *suma = *suma - w*fabs(er);
    else
      *suma = *suma + w*fabs(er);
  }
}
void  determin_extrem (int far *point, int * er1, int *er2, int *er3,
      int *pozm1, int *pozm2, int *pozm3)
{
    round_max(point, er1, pozm1);
    round_max(point, pozm1, er2, pozm2);
    round_next_max(point, *pozm2, er3, pozm3);
}

void round_max(int far *point, int *ermax, int *pozmax)
{
    int poz;
    int poz1;

    poz = *((char*)(point + 60));
    *(ermax) = *((int*)(point + poz));
    *(pozmax) = 0;
    poz1 = poz;
```

```
while(poz1 >= 0)
{
  poz1--;
  if( *(ermax) < *((int*)(point + poz1)) )
  {
    *(ermax) = *((int*)(point + poz1));
    *(pozmax) = poz - poz1;
  }
}
poz1 = 59;
while(poz1 > poz)
{
  if( *(ermax) < *((int*)_(point + poz1)) )
  {
    *(ermax) = *((int*)(point + poz1));
    *(pozmax) = 60 - poz1 + poz;
  }
  poz--;
}
}

void round_next_max(int far *poin, int poz_last_max, int *ermax, int
*pozmax)
{
int poz;
int poz1;

poz = *((char*)(point + 60));
*(ermax) = *((int*)(point + poz);
*(pozmax) = 0;

poz1 = poz;
while(poz1 >= 0 && poz_last_max > (poz - poz1))
{
  poz1--;
  if( *(ermax) < *((int*)(point + poz1)) )
  {
    *(ermax) = *((int*)(point + poz1));
    *(pozmax) = poz - poz1;
  }
}
poz1 = 59;
while(poz1>poz && poz_last_max > (60 - poz1 + poz) )
{
  if( *(ermax) < *((int*)(point + poz1)) )
  {
    *(ermax) = *((int*)(point + poz1));
    *(pozmax) = 60 - poz1 + poz;
  }
  poz--;
}
}

void round_min(int far *point, int poz_last_max, int *ermin, int
*pozmin)
{
    int poz;
    int poz1;

    poz = *((char*)(point + 60));
    *(ermin) = *((int*)(point + poz));
    *(pozmin) = 0;
    poz1 = poz;
```

```
while(poz1 >= 0 && poz_last_max > (poz - poz1) )
{
  poz1--;
  if( *(ermin) > *((int*)(point + poz1)) )
  {
    *(ermin) = *((int*)(point + poz1);
    *(pozmin) = poz - poz1;
  }
}
  poz1 = 59;
  while(poz1>poz && poz_last_max > (60 - poz1 + poz) )
  {
    if( *(ermin) > *((int*)(point + poz1)) )
    {
      *(ermin) = *((int*)(point + poz1));
      *(pozmin) = 60 - poz1 + poz;
    }
    poz1--;
  }
}
```

Pentru a se asigura funcţionarea on-line partea de inferenţă a SE este organizată sub formă de task. Sub controlul executivului lucrează şi alte taskuri cum este cel de comunicaţie pentru supravegherea liniei seriale, cel de tastatură şi afişare. O funcţie task conţine tot mecanismul de inferenţă, funcţie care are structura similară cu cea prezentată în paragraful 10.1.

Două aspecte importante au fost avute în vedere: o strategie de ameliorare a parametrilor de acord în funcţie de dependenţa acestora de mărimile de calitate cît şi modul de integrare a unui sistem expert într-o aplicaţie de timp real. Sistemul a fost testat în condiţii de laborator, rezultatele obţinute fiind promitătoare. În viitor se încearcă perfecţionarea prin determinarea şi a altor dependente între parametrii de acord şi mărimile de calitate.

10.3. Sistem expert pentru diagnoza medicală

Acest exemplu constituie un sistem de diagnosticare în medicină, vizînd detectarea unor maladii comune cum sînt gripă, guturai, viroze. Dezvoltarea de faţă dispune de o interfaţă utilizator simplă cu ajutorul căreia medicul răspunde la întrebări privind starea pacientului. Aprecierea stării este judecata prin valorile funcţiei de apartenenţă în intervalul (–5, 5). Gradul de adecvanţă asociază pentru –5 valoarea absolut false şi pentru 5 valoarea absolut true. Valorile cele mai apropiate de 0 sînt cele mai incerte.

10.3.1. Implementarea mecanismului de inferenţă

```
# include "stdio.h"
# include "stdlib.h"
# include "alloc.h"
```

```
# include "string.h"
# include "math.h"

FILE *fp;
char x[30];
char y[30];
char z[30];
char q[30];
char s1[120];
char s2[120];
char bests[120];
char s3[120];
int i, j, k, ss, best, response, bestvar;
int ipot=0;
int evid=-1;
float *po, *rulevalue, *mini, *maxi;
int *questions, *var, *varflag;
float absolut, p, pp, py, af, pn, pe, rv, maxofmin, prior, a1, a2,
a3, a4;

void main(void)
{
if((fp=fopen("dataa.dat", "r"))==NULL){
printf("cannot open file\n");
exit(1);
                    } /*if open*/
clrscr();
rewind(fp);
fscanf(fp, "%s%s", s1, s1);
while(strcmp(s1, "symptoms")) { /*wh1 desc*/
   fscanf(fp,, "%s%s", s, z);
   while(strcmp(z, "999")){
     fscanf(fp, "%s%s%s", x, y, z);
     }
     ipot++;
     fscanf(fp, "%s", s1);
     }
while(strcmp(s2, "end")){
     fscanf(fp, "%s%s", s2, s3);
     evid++;
     } /*0*/
po=(float *) malloc((ipot + 1) *sizeof(float));
rulevalue=(float *) malloc((evid + 1) *sizeof(float));
questions=(int *) malloc((ipot + 1) *sizeof(int));
mini=(float *) malloc((ipot + 1) *sizeof(float));
maxi=(float *) malloc((ipot + 1) *sizeof(float));
var=(int *) malloc((ipot + 1) *sizeof(int));
varglag=(int *) malloc((evid + 1) *sizeof(int));
if(!po || !rulevalue || !questions || !mini || !maxi || !var ||
!varflag){
     printf("allocation failure\n");
     exit(1);
     }
for(j = 1; j <= (evid + 1); j++){
     varflag[j] = 1;
     rulevalue[j] = 0;
     }
for(j = 1; j <= (ipot + 1); j++){
     mini[j] = 0;
     maxi[j] = 0;
     po[j] = 0;
     questions[j] = 0;
     var[j] = 0;
```

```
    }
/* find the prior probab and rule values */
j = 0;
rewind(fp);
for (k = 1; k <= ipot; k++) questions[k] = 0;
fscanf(fp, "%s", s1);
for(i = 1; i <= ipot; i++){
    fscanf(fp, "%s%s%s", s1, q, y);
    po[i] = atof(q);
    p = po[i];
    j = atoi(y);
    k = 0;
    while(strcmp(y, "999")){
      fscanf(fp, "%s%s", x, z);
      py=atof(x);
      pn=atof(z);
      k++;
      rulevalue[j]+=fabs(p*py/(p*py  +  (1 - p)*pn) - p*(1 - py)  /
      (p*(1 - py)+(1 - p)*(1 - pn)));
      fscanf(fp, "%s", y);
      j=atoi(y);
      }
    var[i] = k;
    questions[i] = k;
    }
/* find the maximum symptom and query */
do {
rv=0;
bestvar=0;
for(j = 1; j <= evid; j++) {
  if (rulevalue[j]>rv) {    /*3*/
    bestvar=j;
    rv=rulevalue[j];
    }
  rulevalue[j]=0;
}    /*1*/
if(bestvar==0) {
  printf("no furthe symptoms\n");
  exit(1);
  fclose(fp);
  }
fscanf(fp, "%s", s1);
for(j = 1; j <= bestvar; j++) fscanf(fp, "%s%s", s2, s3);
printf("\n\nquestions:-\n");
printf("%s", s3);
varflag[bestvar] = 0;

/* obtain user response and adjust probabili */
printf("\nreply on a scale -5(no) to +5(yes) \n");
scanf("%d", &response);
rewind(fp);
fscanf(fp, "%s", s1);
for( i = 1; i <= ipot; i++){
  fscanf(fp, "%s%s", s1, q);
  p=atof(q);
  for( k = 1; k <= var[i]; k++)   {
    fscanf(fp, "%s%s%s", y, x, z); /*AICI*/
    j=atoi(y);
    py=atof(x);
    pn=atof(z);
    if(j!=bestvar  || questions[i]==0) continue;
    questions[i]--;
```

```
      p==po[i];
      pe=p*py + (1 - p)*pn;
      if(response>0) po[i]=p*(1 + (py/pe - 1) *response/5);
      else po[i]=p*(1 + (py - (1 - py)*pe/(1 - pe)) *response/5);
      if(po[i]==floor(po[i])) questions[i]=0;
    }
    fscanf(fp, "%s", y);
  }     /*1*/
/* find new rule values and minimum and max probab */
   rewind(fp);
   fscanf(fp, "%s", s1);
   maxofmin=0;
   best=0;
   for(i = 1; i <= ipot; i++)
     fscanf(fp, "%s%s", s1, q);
     prior=atof(q);
     p=po[i];
     a1=1;
     a2=1;
     a3=1;
     a4=1;
     for(k = 1; k <= var[i]; k++) {
       fscanf(fp, "%s%s%s", y, x, z);
       j=atoi(y);
       py=atof(x);
       pn=atof(z);
       if(varflag[j] *questions[i]==0) continue;
       if(pn>py) {      /*4*/
         py = 1 - py;
         py = 1 - pn;
         }
       rulevalue[j]+=(p*py/(p*py + (1 - p) *pn) - p* (1 - py) / (p*
       (1 - py) + (1 - p)*(1 - pn)));
       a1* = py;
       a2* = pn;
       a3* = (1 - py);
       a4* = (1 - pn);
     }      /*închid for k*/
     fscanf(fp, "%s", y);
     maxi[i]=p*a1/(p*a1 + (1 - p)*a2);
     mini[i]=p*a3/(p*a3 + (1 - p)*a4);
     if (maxi[i] < prior) { /*3*/
       questions[i]=0;
       printf("\nwe can ignoe %s", s1);
       }
     if(mini[i]>maxofmin) {
     best=i;
     maxofmin=mini[i];
       }
}

/* search for a clear winner */
   for(i = 1; i <= ipot; i++) if(mini[best] <= maxi[i] && i!=best)
maxofmin=0;
   }
while (maxofmin==0);
     rewind(fp);
fscanf(fp, "%s", s1);
for( i = 1; i <= best; i++) {
     fscanf(fp, "%s%s", bests, q);
af=atof(q);
for( k = 1; k <= var[i]; k++) {
```

```
        fscanf(fp, "%s%s%s", y, x, z);
        }
     fscanf(fp, "%s", s1);
     }
printf("\n\n the most likelu outcome is %s", bests);
printf("\n with a probability %f", po[best]);
fclose(fp);
free(po);
free(rulevalue);
free(questions);
free(mini);
free(maxi);
free(var);
free(varflag);
}
```

10.3.2. Baza de cunoştinţe

Baza de cunoştinţe este realizată sub forma unui fişier de date ce grupează simboluri pentru diferite simptome. Structura sa este dată mai jos.

```
illenesses
common_cold

1.0             .01          7
1.0             .01          2
1.0             .01          6
1.0             .01         16
1               .01          7
1.0             .01         16
0.0             .5           8
0.0             .01         64
1.0             .01         15
1.0             .01         17
0.0             .01        999 laryngitis
1.0             .01          8
1.0             .01         34
1.0             .01         12
1.0             .01         15
1.0             .01         18
1.0             .01         31
1.0             .01         22
1.0             .01         12
1.0             .01         22
1.0             .01         36
1.0             .01         23
1.0             .01          5
1.0             .01          5
1.0             .01         18
1.0             .01         23
1.0             .01         29
1.0             .9           7
1.0             .01         18
1.0             .01        999
//Sînt date mai jos o serie de simptome

symptoms
1
are-you-snezing-a-lot?
```

```
2
are-your-eyes-painful-or-waterin-a-lot?
3
do-you-have-a-sore-throat?
4
is-your-voice-hoarse?
5
are-you-coughing-a-lot?
6
do-you-have-a-runny-nose?
7
do-you-have-a-headache-or-in-general-do-you-sufer-from-headaches-at-
all?
8
do-you-have-a-high-temperature?
9
do-you-spend-a-lot-of-your-time-in-a-very-dusty-atmosphere?
10
dose-your-skin-itch?
11
do-you-have-a-dry-throat?
12
is-your-breath-"wheezy"?
13
is-your-nose-very-'blocked-up'?
14
have-you-had-a-cold-or-similar-infection-recently?
15
do-you-feel-generally-ill?
16
do-you-have-trouble-swallowing?
17
do-you-muscles-ache?
18
do-you-have-any-pain-at-all-in-your-chest?
19
have-you-had-your-tonsils-removed?
20
do-you-have-any-symptoms-which-tend-to-occur-in-'attacks-'rather-
than-being-present-all-the-time?
21
do-you-have-a-'productive-'cough-a-cough-in-which-you-bring-
something-up?
22
are-you-rather-breathelss?
23
do-you-sweat-a-lot-not-just-when-you-exert-yourself-but-when-you-are-
apparently-relaxing-as-well?
24
is-your-pulse-rate-high-?nomally-it-should-be-about-60-to-80-beats-
each-minute-and-slight-faster-for-people-over-70-or-under-20.
25
do-you-have-severe-attacks-of-breathlessness-enough-to-seriously-
worry-you?
26
does-your-skin-have-a-bluish-tinge?
27
when-you-cough-is-your-phlegm-stained-with-blood?
28
are-you-confused-muddled-about-what's-going-on-around-you?
29
are-you-(or-the-patient)-delirious-talking-incoherently-with-poor
```

```
muscular-coordination?
30
do-you-have-a-dry-(non-productive)-cough?
31
is-it-painful-when-you-breathe-or-cough?
32
do-you-ever-have-any-really-severe-pain-in-your-chest?
33
do-you-swing-between-feeling-chilled-and-feeling-feverish?
34
do-you-have-any-symptoms-wich-have-been-present-for-some-time-
possibly-six-weeks-or-more?
35
do-you-have-'clubbed-fingers'?-these-are-fingers-in-wich-the-
cuticles-have-almost-disappeard-and-nails-curve-over-at-the-
fingertips?
36
do-you-have-any-symptoms-wich-mainly-occur-when-you-exert-yourself?
37
do-you-smoke?-to-reply-divide-the-number-of-cigarettes-you-smoke-by-
5-use-this-number-to-reply-(if-you-smoke-20-a-day-reply-4).5-is-the-
maximum-reply-and--5-means-you-do-not-smoke.
38
do-you-suffer-from-feelings-of-dizziness?
39
do-you-have-palpitations?-the-feeling-that-your-heart-is-beating-
more-strongly-or-faster-or-more-unevenly-than-it-should?
40
is-either-your-ankles-unduly-swollen?
41
are-you-vomiting-or-do-you-have-strong-feelings-of-nausea?
42
do-you-have-any-abdominal-pain?  this-is-pain-anywhere-between-the-
bottom-of-the-tibcage-and-the-groin.
43
do-you-suffer-from-disaphoea?-passing-unusualy-runny-feeces.
44
have-you-your-appendix-removed?
45
do-you-have-jaundice? this-is-not-a-disease-but-a-simptom-of-disease,
often-it-is-most-obvious-in-the-eyes-the-whites-become-yellow.
46
are-you-rather-tense-and-apprehensive?
47
do-you-find-it-hard-to-get-to-spleep-or-do-you-often-wake-in-the-
middle-of-the-night?
48
do-you-have-any-invluntaru-twitching-or-trembling?
49
do-you-suffer-from-constipation?passing-faeces-infrequently-or-with-
difficulty.
50
do-you-have-a-poor-memory? that-is-difficulty-remmebering-individual-
facts-either-occasionally-or-regularly.
51
have-you-totally-or-nearly-lost-the-power-of-speech?
52
have-you-experienced-any-bleeding-from-your-back-passage?
53
are-you-male-or--female?answer-5-for-male-or--5-for-female.if-you'd-
like--the-analysis-to-be-general(for-either-sex)-reply-0.
54
```

```
is-your--neck-stiff-and/or-painful?
55
have-you-sustained-any-kind-of-had-injury-over-the-last-few-weeks?-
even-a-very-slight-injury-can-be-importan.
56
have-you-recently-been-passing-abnormal-looking-faeces-?
57
are-you-passing-large-quatities-wind-either-by-belching-orflatulence?
58
do-you-have-sudden-feeling-of-faintness-feeling-weak-and-unsteady-
maybe-even-losing-counciousness?
59
does-any-part-of-your-body-itch-with-or-without-the-presence-of-any-
rash?
60
do-you-have-a-skin-rash-of-any-sort?
61
is-any-part-of-your-numb-or-do-you-have-tingling-'pins-and-
needles'sensation-anywhere?
62
are-you-overweight-or-underweight? reply-5-for-definite-overweight-
and-5-for-definite-underweight.reply-0-if-your-weight-is-just-right.
63
do-you-have-any-pain-in-your-face-or-forehead?
64
do-you-have-any-swelling-under-the-skin?
65
is-your-urine-abnormaly-coloured?
66
are-you-urinating-unusually-freqently-?
67
is-it-painful-when-you-urinate?
68
is-your-vision-impaired-in-any-way-blurring-or-double-vision-
orseeing-flashing-lights? (this-does-not-include-defects-wich-canbe-
corrected-by-spectacles)
end
```

BIBLIOGRAFIE

[1] **Fuzzy decision trees. In: Fuzzy sets and systems** - *Adamo, J.M.* 4(1980), pp. 207-219.

[2] **Introduction a la conception des systems Intelligents** - *Alexander, J.*, Paris, Massen, 1984.

[3] **Expert Systems Concepts and Examples** - *Alty, J.L., Coombs, M.J.*, N.C.C. Publications, England, 1984.

[4] **Use of expert systems in closed loop feedback control in Proc.** *Arzer, K.E.*, 1986; Amer. Control Conf. Boston M.A. June 1921, 1985.

[5] **Comparison of fuzzy sets on the same decision space. In: Fuzzy sets and systems** - *Baldwin, J.F., Guild, N.C.F.* 2(1979), pp. 213-232.

[6] **Pattern recongnition with fuzzy objective function algorithms** - *Bezdec, James C.* - . New York, Plenum Press, 1981.

[7] **Integration of an Expert System into a Real-Time Software System** - *Beck, Th., Lauber, R.J.* , 1989.

[8] **Expert System Applications** - *Bolc, L., Coombs, M.J.*, Springer-Verlag, 1988.

[9] **L'intelligence artificialle** - *Bonnet, A.*, InterEditions, 1984.

[10] **A computer based consultant for mineral exploration**. *Buda, R. Hart, P.* - Final Report, 1979.

[11] **Implementations of Prolog** - *Campbell, J.A.* New York, John Wiley, 1985.

[12] **L'Anatomie de prolog** - *Caneghem M.V.* , InterEditions, 1986.

[13] **Sisteme de testare a sistemelor cu procesoare de 8 biti**. - *Cârstoiu, D,* Conferinta de microprocesoare, CNETAC, Bucureşti, 1986.

[14] **Sistem expert de autodiagnoză a modulelor multiprocesor.** *Cârstoiu, D.* Sesiunea IRNE Piteşti, 1988.

[15] **Contribuţii la optimizarea proceselor de producţie utilizând sisteme expert în industria chimică** - *Cârstoiu, D.* Teza de doctorat, Institutul Politehnic Bucureşti, iulie, 1989.

[16] **Inference Deficiencies in Rule - Based Expert Systems.** - *Cârstoiu, D.*, 8 - th International Conference on Control Systems and Computer Science, june 1991.

[17] **Asistarea deciziilor clinice prin calculator** - *Cârstoiu, D.*, Revista Academică, Bucureşti, nr. 4, 1991.

[18] **Multi-port Communication between Computers through Radio Connection** - *Cârstoiu, D., Radu, C.*, . Revue Roumanie des Sciences Technique. Serie Electronique et Energetizue, 37, 4, p. 523-530, 1992.

[19] **An Expert Sistem for Improved on line Controller Performance** - *Cârstoiu, D., Radu C.*, . Revue Roumaine des Sciences Technique. Serie Electronique et Energetique, 37, 1, 1993.

[20] **On-line Expert System to Increase the Quality of PID Controllers. IMACS/IFAC Mathematical and Inteligent Models in System Simulation**, *Cârstoiu, D., Radu, C.*, Brussels, Belgium, 1993.

[21] **Parallel Inference Machine Architecture for Rule Based Systems** - *Cârstoiu, D.*, . 9-th International Conference on Control Systems and Computer Science, may 1993.

[22] **Aspects of Physical Database Design and Reorganisation** - *Cârstoiu, D.*, . 9-th International Conference on Control Systems and Computer Science, may 1993.

[23] **Strategii de testare în sistem cu microprocesor.** *Cârstoiu, D., Sandor, V., Algiu, M.*, Conferința de microprocesoare, microcalculatoare şi aplicații, Bucureşti, noiembrie, 1987.

[24] **Sisteme de gestiune a modelelor sistemelor automate** - *Cârstoiu, D.*,. A 4-a Conferința Națională de Electronica, Telecomunicații, Automatică şi Calculatoare, Bucureşti, decembrie, 1988.

[25] **Arhitecturi specifice sistemelor expert cu aplicații în conducerea proceselor. Structuri, algoritmi şi echipamente de conducere a proceselor industriale** - *Cârstoiu, D.*, , Iaşi, p. 333-339, octombrie, 1989.

[26] **Sisteme expert pentru asistarea deciziilor, Structuri, algoritmi şi echipamente de conducere a proceselor industriale** - *Cârstoiu, D.*, , Iaşi, p. 373-379, octombrie, 1989.

[27] **Sisteme expert pentru prelucrarea bazelor de date, Structuri, algoritmi şi echipamente de automatizare a proceselor industriale** - *Cârstoiu, D.*, , Craiova, martie 1991.

[28] **Sisteme suport de decizie cu aplicații în medicină. Structuri, algoritmi şi echipamente de conducere a proceselor industriale,** *Cârstoiu, D.*,Iaşi, p. 101-109, octombrie, 1991.

[29] **Sisteme expert pentru supravegherea calității conducerii.** *Cârstoiu, D.*, SIMSID 7, Galați, p. 261-267, România, 1992.

[30] **Sisteme expert de autodiagnoză în structuri multiprocesor** - *Cârstoiu, D.*, Sesiunea Jubiliară 175 ani de învăţământ, Iaşi, noiembrie 1987.

[31] **Expert Systems in Real Time Control of Technological Processes.** *Cârstoiu, D.*, TITAN '89 Section on Institute for Computers and informatics, Predeal, October, 1989.

[32] **Systemes experts methodes et outils, Eyrolles**, *Chatain, J.N., Dussauchoy, A*1987.

[33] **Programming in Prolog** - *Clocksin, W.F. Mellish, C.S.*, , Springer-Verlag, 1984.

[34] **Explaining induced decision trees. Proceedings Expert Systems** -*Corlett, R.A.*, 1983.

[35] **Sistemi informativi inteligenti** - *Davies, R.* . Padova, France Muzzio Editore, 1988.

[36] **An Expert PID controller in Proc.**, *Devarathan, R.*, IEEE Int. Workshop on Artificial Intelligence for Industrial Applications, 1988.

[37] **Tinning Maps for Three Mode Controllers** - *Dorris M. Wills* , 1962.

[38] **Intelligence artificielle: mythes et limites** - *Dreyfus, H.* Paris, Flammarion, 1979.

[39] **A logic programming metalanguage for expert systems** - *Ernst, CH., J.*, In: Kacprzyk, J., Yager, R. (eds), 1985, pp. 280-288.

[40] **Les SE: pricipes d'organisation et fonctionnement. Informatique et Gestion** - *Farreny, H.* -, no. 142, mai 1983.

[41] **A general model to troubleshooting and its applications to computer support.** - *Frley, A.* 5es journees internationals: les SE - leurs applications, Avignon, 1985.

[42] **La cinquieme generation.** *Feigenbaum, E., McCorduck, P.* - Paris, Inter Editions, 1984.

[43] **Elemente de inteligenţă artificială**. *Georgescu, I.*, Academia Republicii Socialiste România, Bucureşti, 1985.

[44] **PROLOG** - *Giannesini, F.*, Paris, Masson, 1983.

[45] **Uncertainly modele for knowledge based systems. A unified approach to the measurement of uncertainly.** - *Goodman, J.P., Nguyen, N.T.* Amsterdam, Elsevier, 1985.

[46] **Handbook, of Algorithms and Data Structures** - *Gonnet, G.H.*,, Addison-Wesley, 1984.

[47] **Techniques algorithmiques pour l'intelligence artificelle**, *Griffiths, M.* Paris, Hermes, 1986.

[48] **Aproximative reasoning in expert systems**, *Gupta, M., Kandel, A.*, s. a, Amsterdam, 1985.

[49] **Experience in the Use of an Inductive System in Knowledge Engineering. Proceedings of the Fourth Technical Conference of the British Computer Society Group on Expert Systems**, *Hart, A. E.*, 1985.

[50] **Building Expert Systems**, *Hayes, R., Waterman, D.A., Lenat, D.B.*, Addison-Wesley, 1983.

[51] **Intelligent decision support in process environments**, *Hollinagel, E.*, Springer-Verlag, 1086.

[52] **Highly Versatile in Circuit Emulator**, *Ionescu, T.D., Cârstoiu, A. .* 7-th International Conference on Control Systems and Computer Science, may 1987.

[53] **Teoria sistemelor**, *Ionescu, V.*, , Editura Didactică şi Pedagogică, Bucureşti, 1985.

[54] **Theory of fuzzy subsets**, *Kaulmann, A.*,Academic Press, New York, 1965.

[55] **Prolog for programmers**, *Kluzniak, F. Szpakowicz, S.*, Academic Press, 1985.

[56] **Knowledge representation from a knowledge engineering point of view**, *Konrad, W.*, Proceedings of the International Workshop Spa, Belgium, Springer-Verglag, 1986.

[57] **"Self-tunning PID controller uses pattern recongnition approach**, *Kraus, T.W. and T.J. Myron,* Control eng. pag 106-111 June 1984.

[58] **Qualitative, Process Theory,** *Kuipers, B.* Artificial Intelligence Journal, pp. 85-168, North-Holland, 1984.

[59] **An artificial intelligence approach to discovery in Mathematics as heuristic search**, *Lenat, D.*, Memo AIM-206, Stanford AI Laboratory, 1976.

[60] **All about chess and computers**, *Levi, D.*, Berlin, Spinger-Verlag, 1982.

[61] **Applications of Aritificial Intelligence for organic chemistry**, *Lindsay, R.K.*, New York, 1980.

[62] **An Expert System to Performe On-Line Controller Tunning**, *Litt, J.*, American Control Conference, May, 22-25, 1990.

[63] **A theory of syntactic recognition for natural language**, *Marcus, M.P.*, Cambridge, 1980.

[64] **A Knowledge Based Environment for Process Planning**, *Mouleeswarn, C.B., Fischer, H.*, Proceedings of the 1-st International Conference Southaton, Springer-Verlag, 1986.

[65] **Expert Systems and Fuzzy Systems**, *Negoita C.V.*, The Banjamin Cummings Plublishing Company Inc., 1985.

[66] **Concepts of expert systems and languages for their implementation**, *Petzsch, H.*, Proceedings of the International Workshop spa, Belgium, Springer-Verlag, 1986.

[67] **Expert Systems in Engineering**, *Pham, D.T.*, Springer-Verlag, 1988.

[68] **Real Time expert tunners for PI controllers,** *Porter, B., Jones, A. adn C.B. McKeown*. IEEE Pric. D. vol 134, pp. 260-263 July 1987.

[69] **Discovering rules by introduction from large collections of examples**, *Quinlan, J.R.*, In Expert in the Microelectronic Age, Edinburgh, 1979.

[70] **An expert system using palusible inference**, *Reit, J., AL/X:* Oxford: Inteligent, Terminals Ldt, 1980.

[71] **Natural Language inferences**, *Rich, E.*, Auckland, Mc-Graw-Hill, 1984.

[72] **Sisteme expert de autodiagnoză în structuri cu mai multe procesoare**, *V. Sandor, D. Cârstoiu, M. Algiu*, TESS. Revista Automatica, Electronica, Electrotehnica no. 2/1989.

[73] **Process Control Systems**, *Shirskey, F.G.*, - New York, McGraw-Hill, 1988.

[74] **"Applied Optimal Control", Large Scale Systems**, *Singh, M.G.*, Recenzia lucrării 1981.

[75] **Data Structure Techniques**, *Standish, T.A.*, Addison-Wesley, 1980.

[76] **Microprocessors in Signal Processing, Measurement and Control**, *Tzafestas, K.* - Dordrecht D. Reidel, 1983.

[77] **Optimum settings for automatic controllers**, *Ziegler, J.G. and N.B. Nichols* - Trans A.S.M.E. vol 64, pp. 759-768, 1942.

[78] **Fuzzy sets in Decision Making and Expert Systems,** *Zimmermann, H.J.*, Boston, 1987.

[79] **A Guide to Controller Tuning**, *Wills, M.*, IEEE Transaction on Industrial Electronics, vol. 37, oct. 1990.

[80] **Comments on "Optimal gaim for proportional-integral derivative feedback",** *Wolbrot, M. and M.P. Polis* - IEEE Cotrol Syst. Mag. vol. 9, pp. 100-101, June, 1989.

[81] **An Expert System for Real-Time Control IEEE Software**, *Wright I.*, 1986, March, pp. 16-24.

Autorul abordează din diferite unghiuri vastul domeniu al Sistemelor Expert, reuşind astfel să supună atenţiei cititorului o problematică pe cât de complexă pe atât de unitară. Lucrarea surprinde unele fundamente ale Teoriei Sistemelor Expert, dar şi câteva tendinţe de ultimă oră în acest domeniu, venind astfel deopotrivă în sprijinul celor care doresc să se iniţieze, cât şi celor avansaţi în domeniul inteligenţei artificiale.

Diversificata problematică abordată, maniera riguroasă de prezentare şi caracterul unitar impun prezenta lucrare drept una de referinţă în peisajul publicistic de profil.
O recomandăm studenţilor, informaticienilor, medicilor, inginerilor.

ISBN 973-9156-92-1

8950 lei